2017 年度 修士論文

グラフ書換えモデル検査のための差分適用グラフ同型性判定における基本閉路長削減手法

提出日： 2018 年 1 月 26 日
指導： 上田 和紀 教授

早稲田大学 基幹理工学研究科
情報理工・情報通信専攻

学籍番号：5116F040-8
坂爪 裕也

概要

坂爪裕也

モデル検査とはモデル化されたシステムの正当性を検証する手法であり，その検査方法はモデル化されたシステムが取りうる状態を網羅的に探索することで，その状態群が要求された性質を満たすか否かを調査する，というものである．モデル検査が抱える問題として，探索する状態数がモデル自身の大きさに対して爆発的に大きくなる状態空間爆発がある．

状態空間爆発に対する対策として，モデリング言語にグラフ書換え系を用いることで，グラフ構造のもつ対称性吸収機能を利用して状態数の削減を行う，というものが挙げられる．また，グラフ構造の表現力を用いて様々なシステムの検査を実現できることも，モデル検査にグラフ書換え系を用いることの長所である．一方，モデル検査の過程において，構築したグラフ構造が状態空間内に存在するかどうかを頻繁に判定するため，モデリング言語にグラフ書換え系を用いる上で，グラフ構造の効率的な一意表現の算出，および効率的な同型性判定はグラフ書換え系モデル検査器の実用化へ向けての重要な課題である．

そこで本研究では，効率的な判定のために，グラフ書換え系における書換え前後の差分に着目した．適用する書換えが同一であれば書換え差分も同一であるため，書換え時の差分がグラフ構造に占める割合はグラフサイズに反比例して小さくなる．そこで，書換え差分を利用してグラフ同型性判定を行うことで，グラフ同型性判定の効率化を図る．

本論文では，対称差を利用して変化が計算可能なグラフ構造の一意表現を用いたグラフ同型性判定アルゴリズムにおける，状態展開時の対称差の処理量の改善アルゴリズムを提案した．これは，状態空間を状態という頂点と遷移という辺からなるグラフ構造ととらえ，差分情報の計算時に状態空間の全域木を用いることで基本閉路を一意に定め，状態空間内の探索を不要にし，かつ状態空間の展開時に全域木に採用される遷移を更新することで，利用する差分情報の処理量自身も削減する，というものである．同時にこれを実装し，評価を行った．

Abstract

Yuya Sakazume

Model checking is a method of verifying the properties of a modeled system. Model checking exhaustively checks all the states of the modeled system to check whether it meets required properties. However, as the model increases, the number of state to check tends to be very large. This phenomenon is called state space explosion.

We can use symmetry reduction of graph structures to take measures against state space explosion by using graph-based model checking. On the other hand, in the process of graph-based model checking, it is frequently judged whether or not the constructed graph already exists in the state space of the model. So, it is important to efficiently check isomorphism of graphs to realize graph-based model checkers.

As a premise, we make the unique representation of graphs computable with change using symmetric differences. In this thesis, I proposed an algorithm to improve the throughput of difference information during state expansion. Also, I implemented this algorithm and evaluated.

目次

図目次

第 1 章

はじめに

1.1　研究の背景と目的

ハードウェア・ソフトウェアの発展に伴い，必要とされるシステムはより大規模かつ複雑なものになり，安全性や正当性の検証もより困難になっている．にも関わらず情報処理システムに要求される信頼性は年々高まっており，システムの想定外の挙動は甚大な被害をもたらす可能性が高い．

モデル検査はそのようなシステムの正当性・信頼性を保証するための検証手法であり，モデル化されたシステムを状態として状態に対して起こりうる変化を記述し，状態への変化の適用を新たな状態が生成されなくなるまで繰り返し試みることで全ての状態を網羅し，それにより開発者が想定していない仕様外の挙動を起こしうるか，あるいはシステムに要求された性質を満たすか否かなどを検証するものである．モデル検査における代表的な問題として，モデル自身の大きさに対してモデルが取りうる状態数が爆発的に大きくなる状態空間爆発が挙げられる．状態空間爆発に対する対策は状態 1 つあたりのデータ量を圧縮する，状態空間内に保持する状態数を制限する等の手法が代表的であるが，モデリング言語にグラフ書換え系を採用することでグラフ書換え系のもつ高い対称性と対称性吸収機能が状態数の削減に大きな効果を示す．

グラフ書換え系言語をモデリング言語に用いた例として，階層グラフ書換えに基づく言語モデルである LMNtal[6] をモデリング言語とするグラフ書換え系モデル検査器 SLIM[4] がある．SLIM は LMNtal のもつ高い対称性と対称性吸収機能を利用してモデル検査器としての発展を遂げてきた [4, 8] が，LMNtal のみならずグラフ書換え系モデルを利用してモデル検査を行う際，構築したグラフ構造が状態空間内に既にあるかを頻繁に判定する必要がある．これはすなわちグラフ構造の一意表現の算出，および同型性判定を

頻繁に行う必要がある，ということであり，両処理の効率化はグラフ書換え系モデル検査器を実用化する上で避けては通れない課題である．

グラフ構造の正規化はグラフ同型性判定の手法の 1 つである．この手法はグラフ構造の各頂点に一意に対応する整数を割り当てるものであり，正規化されたグラフ同士の同型性判定は割り当てられた整数が等しいグラフ頂点同士の比較のみで可能なことから，グラフ書換え系モデルに対するモデル検査に対しても，グラフ正規化を利用した同型性判定を採用することによる効率化が期待されている．

したがってグラフ書換え系におけるグラフ構造の効率的な正規化は重要な課題であったが，グラフ書換え時においてグラフ構造の変化差分が小さいことが多い事象に注目した，グラフ書換え前のグラフ構造を正規化した際の情報と差分情報を活用してグラフ書換え後のグラフ構造の正規化を行うアルゴリズム [11]，およびグラフの正規化の結果がグラフの書き換え前後において変化が部分的となることに着目した正規化されたグラフ構造の差分を活用した同型性判定アルゴリズム [12] が提案された．

本研究では先述した差分適用グラフ正規化アルゴリズム・差分適用グラフ同型性判定アルゴリズムを実際にグラフ書換え系モデル検査器に導入する過程において必要な，対称差によって計算可能な一意表現と状態空間を用いてグラフ同型性判定を行う際の閉路長を削減する，基本閉路長削減アルゴリズムを提案した．このアルゴリズムの適用による改善効率はモデルがなす状態空間によって大きく変化するため，同アルゴリズムを SLIM に導入し，実際にどの程度の効率化を行えるかを測定した．

1.2　論文の構成

第 2 章では差分適用グラフ正規化アルゴリズム，差分適用グラフ同型性判定アルゴリズム，および基本閉路長削減手法に関連する事項について述べる．具体的には，彩色に関する概念の諸定義や，グラフ構造と全域木に関する諸定義を述べる．

第 3 章では，基本閉路長削減手法を適用する差分適用グラフ同型性判定アルゴリズムについて述べる．具体的には 3.1 節で触れるが，差分適用グラフ正規化アルゴリズムは，それ自身が McKay のグラフ正規化アルゴリズム [2, 9] のグラフ書換え系向けの最適化でありながら，差分適用グラフ同型性アルゴリズムの一部である．

第 4 章では，基本閉路長削減手法に関して述べる．基本閉路長削減手法が差分適用グラフ同型性判定手法のどこに位置するのかに言及したのち，基本閉路長削減手法について具体的に述べる．

第 5 章では実際にグラフ書換え系モデル検査器 SLIM への実装を行い，基本閉路長削

減手法による基本閉路長の削減効率を測定する．また，幅広いモデルに関して効用があるかについても検証する．

第 6 章では本論文のまとめ，および今後の課題について述べる．

第 2 章

諸定義

この章では，3 章以降で触れる差分適用グラフ正規化アルゴリズム，および差分適用グラフ同型性判定アルゴリズムに関する概念の諸定義を述べる．2.1 節から 2.8 節，の定義は文献 [10, 11, 12] に基づく．2.9 節から 2.16 は [5] を参考に，無向木の概念の一部を有向木へと拡張，あるいは新しく定義している．また，2.15 節に関しては更に，[1] を参考にした．

2.1　有限無向単純グラフ

有限無向グラフは有限集合 V $(V \subseteq \mathbb{N})$ と V の成分に関する集合 $E = \{\{u, v\} \mid u, v \in V\}$ の組 (V, E) からなる G をさす．このとき，V をグラフ G の頂点集合，E を G の辺集合と呼ぶ．

有限無向グラフのうち，辺集合 E に両端が同一の頂点である辺 (自己ループ)，両端の頂点の組が同一である複数の辺 (多重辺) を含まないものを有限無向単純グラフと呼ぶ．以降，2.2 節から 2.1 節まで，グラフ G は有限無向単純グラフとして扱う．

2.2　頂点彩色

頂点の彩色とは，グラフ $G = (V, E)$ の頂点集合 V を自然数 $1, 2, \ldots, m$ へ写像する関数 $\varphi : V \longrightarrow \Omega = \{1, 2, \ldots, m\}$ である．ただし，$m \leq n = |V|$ であり，φ は $0 \leq k \leq m$ をみたす整数 k において $\varphi^{-1}(k) \neq \emptyset$ をみたす．

このとき，同一の彩色 i を与えられた頂点の集合 $W_i[\varphi] = \{v \in V \mid \varphi(v) = i\}$ を彩色クラスとよび，$|W_i[\varphi]| = 1$ である彩色クラスを特に一元彩色クラスと呼ぶ．更に，ある

彩色 φ が全単射であるとき，すなわち $1 \leq \forall i \leq n; |W_i[\varphi]| = 1$ のとき，彩色 φ は離散的であるという．

彩色グラフとは，グラフ G とその彩色 φ の組 (G, φ) である．また，頂点集合 V に対する彩色全体の集合を $C(V)$ とする．さらに，有限集合 $V, V' \subseteq \mathbb{N}$，$\varphi \in C(V)$，$v \in V$ のもとで，写像 $g : V \longrightarrow V'$ は $\varphi^g(v) = \varphi(v^{g^{-1}})$ であるとする．

2.3　同型性と正規形

$K(V)$ を，頂点集合に V $(V \subseteq \mathbb{N})$ を持つグラフ全体の集合とする．また，2 つのグラフ $G = (V, E), G' = (V', E')$ $(G \in K(V), G' \in K(V'))$ の頂点間の写像 $g : V \longrightarrow V'$ について，G, G' 間の辺集合の写像 $h : \{u, v\} \in E \longrightarrow \{g(u), g(v)\} \in E'$ を定める．

2 つのグラフ G, G' が同型であるとは，写像 $g \cdot h$ の両方が全単射となること，すなわち G に含まれるすべての頂点および辺が G' に含まれる頂点および辺と一対一で結びつくような状態である，ということである．G, G' が同型であることを $G \cong G'$ と表記し，このときの写像 g を G, G' 間の同型写像と呼ぶ．ここで，頂点集合 V 内の頂点 v どうしを置換する写像の集合，すなわち V の対称群を $Sym(V)$ として，$G \in K(V)$ の自己同型群 $Aut(G)$ は $Aut(G) = \{g \in Sym(V) | G^g = G\}$ である．

正規形とは任意の $G \in K(V)$ および全単射 $g' : V \longrightarrow V$ に対し，

(i) $CF(G) \cong G$
(ii) $CF(G^{g'}) = CF(G)$

の 2 条件を満たす写像 $CF : K(V) \longrightarrow K(V)$ とする．

定義から，2 つのグラフ G, G' において，G と G' が同型であれば正規形は同一である．すなわち，$G \cong G' \Leftrightarrow CF(G) = CF(G')$ である．

ここで，同型性と正規形の概念を，彩色グラフへ適用できるように拡張する．

$K(V) \times C(V)$ を頂点に V を持つ彩色グラフ全体の集合とする．2 つの彩色グラフ $(G, \varphi), (G', \varphi')$ $((G, \varphi) \in K(V) \times C(V), (G', \varphi') \in K(V') \times C(V'))$ が同型であるとは，G, G' 間の写像 $g : V \longrightarrow V'$ が全単射であり，かつ g によって $G' = G^g, \varphi' = \varphi^g$ となることであり，$(G, \varphi) \cong (G', \varphi')$ と表記する．このとき，写像 g を $(G, \varphi), (G', \varphi')$ 間の（彩色を保存する）同型写像と呼ぶ．$G^g = G, \varphi^g = \varphi$ をみたす $g \in Sym(V)$ の集合が (G, φ) の自己同型群および彩色保存同型群であり，$Aut(G, \varphi)$ と表記する．

彩色グラフにおける正規形とは，任意の $G \in K(V)$，$\varphi \in C(V)$，全単射 $g' : V \longrightarrow V$，および $h \in Sym(V)$ に対し，以下の 3 条件を満たす写像 $CF : K(V) \times C(V) \longrightarrow$

$K(V) \times C(V)$ である．

(i) $CF(G,\varphi) \cong (G,\varphi)$
(ii) $CF(G^{g'},\varphi^{g'}) = CF(G,\varphi)$
(iii) $CF(G^{g'},\varphi^{g'}) = CF(G,\varphi) \Rightarrow \exists h \in Aut(G), \varphi^{g'} = \varphi^{h}$

したがって，2 つの彩色グラフ $(G,\varphi),(G',\varphi')$ において，$(G,\varphi) \cong (G',\varphi') \Leftrightarrow CF(G,\varphi)) = CF(G',\varphi')$ である．

2.4　頂点の等価性

グラフ $G=(V,E)$ 内の $v,w \in V$ が G 上で等価であるとは，G における同型写像 g によって $v^g = w$ となることとする．また，彩色グラフ $(G,\varphi) = ((V,E),\varphi)$ 内の $v,w \in V$ が (G,φ) 上で等価であるとは，(G,φ) における同型写像 g によって $v^g = w$ となることとする．

2.5　頂点分類

頂点集合 V における彩色 $\varphi,\varphi' : V \longrightarrow \Omega$ に関して，$\forall u,v \in V\ [\varphi(u) < \varphi(v) \Rightarrow \varphi'(u) < \varphi'(v)]$ であるとき，彩色 φ' は φ の細分化であるとする．すなわち，φ' は φ において異なる色が割り当てられていた頂点同士に異なる色を割り当てたまま，同じ色が割り当てられていた頂点のうち 0 個以上に別の色を割り当てたものである．

彩色グラフ (G,φ) に対して，各頂点 $v \in V$ の隣接頂点 ω に割り当てられた彩色 $\varphi(\omega)$ の集合 $\varphi_{adj}(v) = \{\ \varphi(\omega) \mid \omega \in adj(v)\ \}$ の要素を辞書式順序に並び替えて比較し，順序付けた際の番号を彩色とすることで各彩色クラスの多重集合 $\varphi_{adj}(v)$ の等しいものどうしへの分割が可能であり，この彩色を φ の細分化とできる．この細分化方法を $R : \Omega \longrightarrow \Omega$ とする．

$\varphi^{(0)} = \varphi,\ \varphi^{i+1} = R(\varphi^{(i)})(i > 0)$ とすれば，$\varphi^{(r)} = \varphi^{r+1}$ となる r が $r < n = |V|$ において必ず存在する．これをみたす最小の r を r_0 として，$\varphi^{(r_0)}$ を $\overline{\varphi}$ と表記し，φ の安定細分化と呼ぶ．また，$\varphi = \overline{\varphi}$ であるとき，φ は安定であるという．

2.6　n-近傍グラフ

McKay のグラフ正規化アルゴリズム [2, 9] では，構造の解析を隣接する頂点への情報の伝播の繰り返しによって行なっている。伝播した回数によってある頂点からどの程度離れた頂点の情報までが伝わっているのかが変わるため，伝播の回数に対する範囲を表現する概念を文献 [11, 12] で定義している．

グラフ $G = (V, E)$ 内における $v, w \in V$ 間の距離を G 内における v と w をつなぐ最短路に含まれる辺の個数とし，$\partial(v, w)$ と表記する．ただし，v と w が同一の連結成分に属していない場合，すなわち v, w が異なる連結成分に含まれる場合は $\partial(v, w) = \infty$ とする．また，頂点 v から距離 n 以内の頂点を $D_n(v) = \{w \in V \mid \partial(v, w) \leq n\}$ とする．さらに，頂点 v の n-近傍グラフ $S_n(v)$ を，頂点 v から距離 n 以内の頂点 $D_n(v)$ を頂点集合 $\mathcal{V}(S_n(v))$ に，両端が $D(v)$ に含まれる辺を辺集合 $\mathcal{E}(S_n(v))$ に持つグラフと定める．このとき，$\mathcal{E}(S_n(v))$ は辺の少なくとも片方の端点が $D_{n-1}(v)$ に含まれることから，

$$\mathcal{E}(S_n(v)) = \begin{cases} \emptyset & (n = 0) \\ \{e \in E \mid e \cap D_{n-1}(v) \neq \emptyset\} & (n > 0) \end{cases}$$

とも定めても同義である．

2.7　拡張隣接

文献 [11, 12] において，n-近傍グラフを深さ n の深さ優先探索で探索した際に得られる情報を表現する概念を定めている．ただし，深さ優先探索中にある頂点に到達した際，その頂点が既に探索されているかを考慮しない．

彩色グラフ $((V, E), \varphi)$ における，頂点 v の n 有界拡張隣接関係 $EC_n(v)$ を以下のように定める．

$$EC_n(v) = \begin{cases} \varphi(v) & (n = 0) \\ (\varphi(v), \{\!|EC_{n-1}(w) \mid w \in adj(v)|\!\}) & (n > 0) \end{cases}$$

ここで，$\{\!|...|\!\}$ は多重集合である．また，頂点 v の拡張隣接関係を，$EC_\infty(v) = EC_{|V|}(v)$ と定める．

ある n において $EC_n(v) \neq EC_n(w)$ であった場合，$m > n$ をみたすような m においても $EC_n(v) \neq EC_n(w)$ である．したがって，グラフによっては $n < |V|$ であるような n で彩色グラフ内の全ての頂点の区別がつく可能性がある．

頂点 v, w が等価ならば $EC_\infty(v) = EC_\infty(w)$ である．しかし，この逆は成立しない．すなわち，$EC_\infty(v) = EC_\infty(w)$ となる v, w は等価でない可能性がある．

また，$EC_n(v)$ の定義より，$EC_i(v)$ に含まれる全ての頂点の彩色は n-近傍グラフ $S_i(v)$ 内の頂点の彩色に含まれる．すなわち，$EC_i(v)$ は $S_i(v)$ のグラフ構造と各頂点の彩色により一意に定まる．

2.8　書き換え距離

グラフ書換えは，グラフ構造の変化を頂点および辺の追加・削除によって行う．n-近傍グラフ内の頂点および辺にグラフ書換え処理による変化が起きたか起きていないかを判断するために，頂点 v が書換え処理を受けた箇所からどれだけ離れているかを表す概念を文献 [11, 12] において定めている．

グラフ書換えにおける書換え前の彩色グラフ $((V_1, E_1), \pi_1)$ と書換え後の彩色グラフ $((V_2, E_2), \pi_2)$ に対して，頂点 $v \in V_2$ の書換え距離 $rwd(v)$ を以下のように定める．

(i) $v \in V_2 \setminus V_1 \longrightarrow rwd(v) = 0$
(ii) $v \in V_2 \cap V_1$ かつ $\pi_1(v) \neq \pi_2(v) \longrightarrow rwd(v) = 0$
(iii) $v \in V_2 \cap V_1$, $\pi_1(v) = \pi_2(v)$ かつ $E_1 \triangle E_2$ に v を端点に持つ辺が存在するとき $rwd(v) = 1$
(iv) (i)，(ii)，(iii) により rwd が定まる頂点の集合を V' として，$v \in V_2 \setminus V'$ について，$\partial(v, w)$ をグラフ (V_2, E_2) 上での距離として $rwd(v) = \min\{rwd(w) + \partial(v, w) \mid w \in V'\}$

ここで $n < rwd(v)$ とすると，$S_n(v)$ は $\mathcal{V}(S_n(v)) = \mathcal{V}(G_1) \cap \mathcal{V}(G_2)$，$\mathcal{E}(S_n(v)) = \mathcal{E}(G_1) \cap \mathcal{E}(G_2)$ であり，書換えの前後で変化しないことがわかる．

前述のとおり，次の節から，グラフは無向単純グラフを指すとは限らない．そのため，グラフという用語ごとに，どのグラフを対象としているか但し書きを付している．

2.9　(有限) 有向グラフ

有限無向グラフは有限集合 V $(V \subseteq \mathbb{N})$ と V の成分に関する組の集合 $E = \{\{u, v\} \mid u, v \in V\}$ の組 (V, E) からなる G をさすが．辺集合 E の定義を $A = \{(u, v) \mid u, v \in V\}$ と順序対にし，有向辺としたものを有向グラフという．このとき，有向辺 $a = (u, v)$ に対

し，u を a の始点，v を a の終点という．無向グラフと同様，有向グラフ $DG = (A, V)$ のうち，V を DG の頂点集合，A を DG の有向辺集合と呼ぶ.

また，有向グラフ $DG = (A, V)$ について，DG と同一の頂点集合 V を持ち，有向辺集合 A に含まれる全ての順序対 $a = (v, w)$ を 2 頂点の組 $e = \{v, w\}$ へと変換した組の集合 E からなる，無向グラフ $G = (E, V)$ を考える．このとき，有向グラフ DG を無向グラフへと捉えなおしたものが G である，と定義する．

2.10　閉路

有向グラフ $DG = (A, V)$ に属する辺 $a \in A$ を $a = (v_{src}, v_{dst})$ とおく．

このとき，$0 \leq i \leq n-1$ を満たす i において $v_{i_{dst}} = v_{i+1_{src}}$ を満たすような有向辺の系列 $DP = (a_0, a_1, ..., a_n)$ を (有向) 路という．$v_{0_{src}}$ を DP の始点，$v_{n_{dst}}$ を DP の終点という．また，a を始点，b を終点とする路が存在するとき，a, b 間の路を $DP(a, b)$ と表す．更に，始点と終点が等しい路を閉路という．

同様に，無向グラフにおける路および閉路をおく．すなわち，無向グラフ $G = (E, V)$ に属する辺を $e_i = \{v_{i_{src}}, v_{i_{dst}}\}$ としたとき，$0 \leq i \leq n-1$ を満たす i において $v_{i_{dst}} = v_{i+1_{src}}$ を満たす辺の部分集合 $P = \{e_0, e_1, ..., e_n\}$ を路，e を始点，f を終点とする路が存在するときの e, f 間の路を $P(e, f)$，路のうち始点と終点が等しい路を閉路という．ただし，無向グラフにおける辺 e は頂点の組であることに注意する．

なお，議論を簡単にするため，当論文においては，有向グラフ・無向グラフに関わらず閉路は同じ頂点を 2 度通らないことを前提とする．

2.11　全域木

無向グラフ G において，辺を 1 本以上持つような閉路が存在せず，かつ任意の 2 頂点 $u, v \in V$ に対してある 1 点 u が始点・もう 1 点 v が終点となるような路が存在する場合，その無向グラフは木であるという．

木は閉路を持たないその定義上，任意の 2 頂点に対して路が一意に定まる．

また，グラフ $G = (E, V)$ と木 $T = (F, W)$ に対し，$W = V \wedge F \subseteq E$，すなわち G と T が同じ頂点集合を持ちながら T が木となりうるような E の部分集合を辺集合として持っているとき，T は G の全域木であるという．G において任意の 2 頂点間の路が少なくとも 1 つ存在すれば，G に対応する全域木 T は存在する．

2.12　無向木における距離と直径

2.11 にて言及したが，無向木 $T = (F, W)$ の頂点集合 W に属するある 2 頂点 v, w の間には，必ず一意に定まる路 $P(v, w)$ が存在する．このとき，$|P(v, w)|$ を v, w 間の距離といい，$dist\{v, w\}$ と表す．木内における 2 頂点の距離の最大値，すなわち $max\{dist\{v, w\}, v, w \in W\}$ を無向木 T の直径という．

2.13　全域有向木

有向グラフ $DG = (A, V)$ において，有向辺を 1 本以上持つような閉路が存在せず，特定の頂点 $r \in V$ から他の任意の頂点 $v \in V \setminus r$ に対しての路が存在するとき，その有向グラフは有向木であるという．有向木も無向木と同様閉路を持たないので，$DG = (A, V)$ を無向グラフへと捉えなおしたグラフ $G = (E, V)$ は無向木である．このとき，特定の頂点 r を根という．

また，有向グラフ $DG = (A, V)$ と有向木 $DT = (B, W)$ に対し，$W = V \wedge B \subseteq A$, すなわち DG と DT が同じ頂点集合を持ちながら DT が有向木となりうるような A の部分集合を辺集合として持っているとき，DT は DG の全域有向木であるという．無向グラフにおいては全域木の存在条件は任意の 2 頂点間の路が少なくとも 1 つ存在することであったが，有向グラフにおける全域有向木の存在条件は

- $A = (A, V)$ 内の全ての辺で終点とされていない頂点 $r \in V$ がただ 1 つ存在する
- r から他の任意の頂点 $v \in V \setminus r$ への路が少なくとも 1 つ存在すること

である．

有向木 $DT = (B, W)$ 内に有向辺 $b = (v, w) \in B$ が存在するとき，v を w の親，w を v の子であるという．

2.14　深さ

無向木同様，有向木 $DT = (B, W)$ において，根 r と r 以外の任意の頂点 $v \in V \setminus r$ との間の路 $DP(r, v)$ は必ず一意に定まる．このとき，$|DP(r, v)|$ を頂点 v の深さという．

路を持つ頂点とは別に，あらかじめ r の深さは 0 と定める．

2.15　最小共通祖先と有向木における距離，直径

有向木 $DT = (B, W)$ において，根 r からある 2 頂点 $v, w \in W$ への有向路 $DP(r, v), DP(r, w)$ を考える．このとき，有向路 $DP(r, v) \cap DP(r, w)$ の終点，すなわち $DP(r, v), DP(r, w)$ を辿って r から v, w のそれぞれへと向かう際，初めて分かれる有向辺の始点を v, w の最小共通祖先という．

また，v, w の最小共通祖先が x であるとき，有向木における任意の 2 頂点間 v, w の距離を $|DP(x, v)| + |DP(x, w)|$ と定める．ここで，有向木 DT を無向グラフへと捉え直した無向木 T を考えると，最小共通祖先 x の定義から

$$P(r, v) \cap P(r, w) = P(r, x)$$

であり，

$$\begin{aligned} p(v, w) &= (P(r, v) \cup P(r, w)) \setminus P(r, x) \\ &= P(x, v) \cup P(x, w) \end{aligned}$$

である．よって，無向木 T においても v, w 間の距離は $dist\{v, w\} = |P(x, v)| + |P(x, w)|$ であるので，有向木における v, w 間の距離も無向木における v, w 間の距離同様，$dist(v, w)$ で表すものとする．

2.16　基本閉路・基本閉路長

無向木 $T = (F, W)$ に無向辺 $\{(w, v)\}$ を追加すると，T 内にただ 1 つ，$P(v, w) \cup \{\{w, v\}\}$ からなる閉路を含む．これを T と $\{(w, v)\}$ からなる基本閉路 $FC(v, w)$ と呼び，$|FC(v, w)| = |P(v, w)| + 1$ を $FC(v, w)$ の基本閉路長と定義する．

また，基本閉路・基本閉路長の概念を有向木に拡張する．有向木 $DT = (B, W)$ に有向辺 (v, w) を追加すると，頂点 w の 1 頂点のみ，DT 内の v, w の最小共通祖先 x からの路を

- (v, w) の追加前から存在していた路 $DP(x, w)$
- (v, w) の追加後にできた路 $DP(x, v) + (v, w)$

2 つ持った状態になる．2.10 節における有向グラフの閉路の定義には当てはまらないが，以降の章での議論のために，上の 2 つの路の集合 $\{DP(x, w), (DP(x, v) + (v, w))\}$

を DT と (v, w) からなる基本閉路 $DFC(v, w)$ と定義する．また，$|DFC(v, w)| = |DP(x, w)| + |DP(x, v)| + 1$ を $DFC(v, w)$ の基本閉路長と定義する．

2.17　状態空間

グラフに “グラフを書き換える何らかの概念” を追加した系，すなわちグラフ書換え系を考える．ここでは，書き換える概念を暫定的に書換えルールと呼ぶ．グラフを書換えルールによって書き換えていくことで，グラフ書換え系によって表現されたモデルの変化を表現することができる．書換えを行った際の構造のそれぞれを状態，書換えによる状態の移り変わりのそれぞれを遷移と呼ぶ．

グラフ書換え系が 1 つの状態から複数の状態に遷移することが可能であるように設計されている場合，書換えルールによって書換え可能な状態がなくなるまで，初期状態から何らかの順番で各状態に書換えルールを適用していくことで，グラフ書換え系によって表現されたモデルが取りうる全ての状態および遷移を構築することができる．このとき，構築中か構築後であるかに関わらず，その時点までのモデルがとった状態と遷移の集合を状態空間と呼ぶ．すなわち，状態空間は，状態という頂点と遷移という有向辺からなるグラフ構造を指す．よって，有向木における議論は，そのまま状態空間に適用することが可能である．

第 3 章

差分適用グラフ同型性判定アルゴリズム

この章では基本閉路長削減手法を適用する，差分適用グラフ同型性判定アルゴリズムについて述べる．

また，本章におけるグラフに関する議論は，有限無向単純グラフを対象として行う．

3.1　差分適用グラフ同型性判定アルゴリズムの概要

両アルゴリズム本体に関する記述の前に，両アルゴリズムの理解の一助とするため，先に差分適用グラフ同型性判定アルゴリズムの概要，および『差分適用グラフ同型性判定アルゴリズム』と『差分適用グラフ正規化アルゴリズム』の間の関係について述べる．

差分適用グラフ同型性判定アルゴリズムは，グラフの書換え差分からグラフの各情報の差分を段階的に算出していくことで，グラフ書換え系モデル検査における状態空間の効率的な構築を図るアルゴリズムである．このアルゴリズムは，おおまかに以下の 4 ステップからなる．各概念に関しては，のちにそれぞれの節で触れる．

(i) 書換え前のグラフ，書換え前のグラフの正規化結果，および書換え後のグラフを利用して，書き換え後の正規化結果を得る．
(ii) 書換え前のグラフの一意表現，書換え前のグラフの正規化結果と書換え後のグラフの正規化結果の差分から，書換え後のグラフの一意表現と書換え前後の一意表現の対称差を得る．
(iii) 書換え後のグラフの一意表現を用いて，状態空間内の同一状態候補を検索する．

(iv) 候補が存在すれば，書換え前後の一意表現の対称差，および状態空間内の各状態の一意表現の対称差を用いて，書換え後のグラフと同一状態候補の一意表現の対称差を計算する．

上記の 4 ステップのうち，差分適用グラフ正規化アルゴリズムは (i) のステップを担うものである．処理の詳細は 3.3 節，特に 3.3.2 にて述べる．また，3.4.6 にて，上記の (ii) の処理方法に関して述べている．

3.2　差分適用グラフ正規化アルゴリズム

この節では彩色単純グラフにおけるグラフ正規化手法である McKay のグラフ正規化アルゴリズム [2, 9] の概要およびグラフ書換え系への最適化を行う箇所に関する性質について述べたのち，差分適用グラフ正規化アルゴリズム自身に関して述べる．

3.2.1　McKay のグラフ正規化アルゴリズム

本節では，差分適用グラフ正規化アルゴリズムにおいてグラフ書換え系向けに最適化するもととなる，McKay のグラフ正規化アルゴリズム [2, 9] の概要について述べる．また，差分適用グラフ正規化アルゴリズムにおいて利用している，McKay のグラフ正規化アルゴリズムにおいて成立する性質についても言及する．

3.2.2　アルゴリズムの概要

McKay のグラフ正規化アルゴリズムは，与えられた彩色グラフ $((V, E), \varphi)$ に対し，φ の細分化を繰り返すことで離散的彩色を算出し，グラフ構造 (V, E) に対して一意な離散的彩色を得るというものである．このアルゴリズムでは，彩色の値の順に頂点を並び替えた隣接頂点の行列を辞書式で比較し，最大となるものを一意的な彩色としている．φ の一意な離散的彩色を得ることで，容易に一意なグラフ構造の表現を生成することができる．

McKay のアルゴリズムでは，φ に対して (i),(ii) の操作を繰り返し，φ の一意な離散的彩色を得る．ただし，最初に与えられた φ は安定細分化ではないものとする．

(i) 現在の φ の安定細分化を得る．

(ii) 安定細分化された φ が離散的であれば操作を終了する．φ が離散的でなければ，複数の頂点が属する彩色のうち最小の彩色を持つ頂点の中から一つ選び，選んだ頂点とこの彩色を持つ他の頂点とを別の彩色に分けることで彩色を細分化する．

差分適用グラフ正規化アルゴリズムでは，(ii) において文献 [2] にしたがって複数のグラフ頂点が属する色のうち最小の色を対象としているが，同一の彩色に対して返す安定細分化が常に一定である操作であるならば，得られる離散的彩色の集合は一意に定まる．また，(ii) の頂点の選択は複数候補が存在するため，深さ優先探索を行う．探索を行なった結果生成された 1 つ以上の離散的彩色の中から，彩色の値の順に頂点を並び替えた隣接頂点の行列を辞書式で比較し，最大となるものを一意的な彩色としている．

なお，McKay のグラフ正規化アルゴリズムは (ii) において生成された離散的彩色どうしの比較を行い，その結果得られたグラフの自己同型群を用いた探索木の枝刈りを行うが，差分適用グラフ正規化アルゴリズムにおいて主たる部分ではないため本論文では省略する．

3.2.3　グラフ書換え系への最適化

この節では McKay のグラフ正規化アルゴリズムのうち，差分適用グラフ正規化アルゴリズムに関連する定理を紹介する．この節の内容は文献 [12] に基づく．

定理 3.2.3.1 は安定細分化を得る過程において，有限拡張隣接による情報が彩色の細分化に反映されていることを示すものである．

定理 3.2.3.1　(X, φ) を彩色グラフとする．このとき，

$$\forall k \in \mathbb{N},\ \forall v, w \in \mathcal{V}(X)\ [EC_k(v) = EC_k(w) \Leftrightarrow \varphi^{(k)}(v) = \varphi^{(k)}(w)]$$

となる．

また定理 3.2.3.2 は，一定以上の深さの有界拡張隣接が無限の深さの拡張隣接に対応する十分な情報を持っていることを表すものである．

定理 3.2.3.2　(X, φ) を彩色グラフとする．また，$|\mathcal{V}(X)| = n$ とする．このとき，

$$\forall v, w \in \mathcal{V}(X)\ [EC_\infty(v) = EC_\infty(w) \Leftrightarrow \forall m \geq n\ [EC_m(v) = EC_m(w)]]$$

となる．

定理 3.2.3.1 および定理 3.2.3.2 から，系 3.2.3.1 が示される．これは拡張隣接による情報が安定細分化で得られる彩色に反映されていることを示すものである．

系 3.2.3.1　(X, φ) を彩色グラフとして，

$$\forall v, w \in \mathcal{V}(X)\ [EC_\infty(v) = EC_\infty(w) \Leftrightarrow \overline{\varphi}(v) = \overline{\varphi}(w)]$$

となる．

したがって，彩色グラフ内の各頂点 v の安定細分化 $\overline{\varphi}$ は，v が属する連結成分全体 $D_n(v)$ の影響を受けている．

定理 3.2.3.3 は，n-近傍グラフ S_i によってのみ各頂点同士の彩色 $\varphi^{(i)}$ における大小関係が定まることを示すものである．

定理 3.2.3.3　彩色グラフ $(X_1, \varphi_1), (X_2, \varphi_2)$ について，

$$\begin{aligned}&\forall v_1, w_1 \in \mathcal{V}(X_1),\ \forall v_2, w_2 \in \mathcal{V}(X_2)\\ &[(S_i(v_1), \varphi_1) \cong (S_i(v_2), \varphi_2) \wedge (S_i(w_1), \varphi_1) \cong (S_i(w_2), \varphi_2)\\ &\qquad \Rightarrow (\varphi_1^{(i)}(v_1) \leq \varphi_1^{(i)}(w_1) \Leftrightarrow \varphi_2^{(i)}(v_2) \leq \varphi_2^{(i)}(w_2))]\end{aligned}$$

となる．

上の定理より，系 3.2.3.2 が示される．この系は有界拡張隣接関係 EC_i が n-近傍グラフ S_i によってのみ定まるというものである．

系 3.2.3.2　彩色グラフ $(X_1, \varphi_1), (X_2, \varphi_2)$ について，

$$\forall v_1 \in \mathcal{V}(X_1),\ \forall v_2 \in \mathcal{V}(X_2)\ [(S_i(v_1), \varphi_1) \cong (S_i(v_2), \varphi_2) \Rightarrow EC_i(v_1) = EC_i(v_2)]$$

となる．

系 3.2.3.2 により，McKay のグラフ正規化アルゴリズムにおいて，初回の安定細分化の過程における $\varphi^{(i)}$ は，n-近傍グラフ $S_i(v)$ の構造によってのみ定まり，書換えが行われたとしても $S_i(v)$ が書き換え前後で等しい．すなわち，$S_i(v)$ に書き換え箇所が含まれていなければ書き換え前後での細分化時の大小関係は変わらない．

そこで，グラフ書換え前の構造に McKay のグラフ正規化アルゴリズムを適用する際，各頂点 v に対する n-近傍グラフ $S_n(v)$ の構造情報を保持しておき，グラフ書換え後の構造の McKay のグラフ正規化アルゴリズムの初回安定細分化時にこれらを利用することで時間計算量の改善を図ることができる．次の節で言及する差分適用グラフ正規化アルゴリズムは，以上のアイディアから考案されたアルゴリズムである．

なお，初回の安定細分化の処理より後における彩色の情報はその頂点が属する連結成分全体の構造に依存し，グラフ書換え前後で使用される情報が異なるため，情報の再利用は難しいと考えられる．

3.3　差分適用グラフ正規化アルゴリズム

本節では，McKay のグラフ正規化アルゴリズムに，グラフ書換え前の n-近傍グラフ $S_n(v)$ と細分化の進行の情報の保持，およびグラフ書換え後の初回安定細分化時の保持した情報の利用を行うことで実行効率の改善を図った，差分適用グラフ正規化アルゴリズムに関して述べる．

なお，この節の内容は文献 [11, 12] に基づく．

3.3.1　n-近傍グラフの情報の保持

差分適用グラフ正規化アルゴリズムでは，グラフ書換え前の初回安定細分化処理時に n-近傍グラフ $S_n(v)$ の代わりに $S_n(v)$ から得られる有界拡張隣接関係 $EC_n(v)$ の値をハッシュ値で保存し，全ての $EC_n(v)$ のハッシュ値をトライ木で管理することで書換え後の初回安定細分化時に情報の再利用を可能としている．本節ではこれらの構造に関して紹介する．

トライ木による有界拡張隣接関係の保持

有界拡張隣接関係 $EC_n(v)$ のうち，$n \geq rwd(v)$ であるものはグラフ書換えの前後で変化する可能性がある．逆に，$n \leq rwd(v)$ である $EC_n(v)$ はグラフ書換えの前後で変化しないので再利用が可能である．$rwd(v)$ の値に関わらず再利用できる EC_n を全て利用するために，また複数の頂点で共通となる $EC_n(v)$ を効率的に保持するために，差分適用グラフ正規化アルゴリズムでは初回安定細分化までに得られる全ての $EC_n(v)$ をトライ木で保持，管理している．

トライ木の根から子ノードへの枝は $EC_0(=\varphi)$ をラベルとしており，以下深さ n+1 の各ノードが EC_n に対応する．初回安定細分化が終了したときのグラフの各頂点はトライ木の葉に保持される．ここで，$EC_\infty(v)$ が同一である頂点は初回安定細分化の際，同一の彩色クラスに分類されてしまうが，その場合は複数の頂点がトライ木の同一の葉に保存される．また，$EC_n(v)$ によって一元彩色クラスに属することがわかった頂点 v に対しては，$m > n$ である $EC_m(v)$ を計算せずとも v が一元彩色クラスに属することが明らかであるので，一元彩色クラスに属することがわかったグラフ頂点に関してはそれ以降の $EC_m(v)$ を計算しない．以上の 2 点と細分化が有限回しか進行しないことより，細分化を表現するためのトライ木の深さは有限で十分であることがわかる．

”$EC_n(v)$ を計算する”と述べた理由に関しては次の節にて述べる．

有界拡張隣接関係のハッシュ値

トライ木に全ての $EC_n(v)$ の構造を直接保持することは非効率的である．そこで差分適用グラフ正規化アルゴリズムでは，$EC_n(v)$ をハッシュ値化し，ハッシュ値を各ノードに保持することで比較の際の効率を大幅に向上させている．

McKay のグラフ正規化アルゴリズムにおいては，比較した順序を彩色としていることからもわかるとおり，連続する自然数を彩色に用いている．だが，彩色は各頂点の分類のためにつけた仮の分類番号であるため，もし彩色が飛び飛びの値であったとしても彩色がグラフ構造に影響を与えることは一切ない．そこで，差分適用グラフ正規化アルゴリズムでは，$EC_n(v)$ をハッシュ値化した上で，$EC_n(v)$ のハッシュ値を細分化後の各頂点 v の彩色としている．ただし，一元彩色クラスに属することが判明した頂点に関しては，判明した際の $EC_n(v)$ をそれ以降の細分化時の彩色としている．これにより，McKay のグラフ正規化アルゴリズムでは比較時の頂点の順序に依存していた，すなわちグラフ全体の構造に依存していた彩色が当アルゴリズムでは v の EC_n のみに依存するようになり，書換え前後での彩色の再計算を最小限に抑えられるようになっている．

EC_n に基づいてハッシュ値を計算する関数，$ECHsh^{(n)} : \mathcal{V}(X) \longrightarrow \mathbb{N}$ は以下のように定められている．ただし，$setHsh : \mathbb{N} \times \{X \in \mathbb{N}^{\mathbb{N}}\} \longrightarrow \mathbb{N}$ は 1 つの整数と有限のハッシュ値の多重集合からハッシュ値を算出するハッシュ関数であるとする．

$$ECHsh^{(n)}(v) = \begin{cases} \varphi(v) & (n = 0) \\ setHsh(\varphi(v), \{\!| \ ECHsh^{(n-1)}(w) \mid w \in adj(v) |\!\}) & (n > 0) \end{cases}$$

$ECHsh^{(n)}(v)$ は $EC_n(v)$ から一意に定まるものである．系 3.2.3.2 より，$EC_n(v)$ は n-近傍グラフ $S_n(v)$ から一意に定まるため，$ECHsh^{(n)}(v)$ は $S_n(v)$ に依存する．すなわち，書換え後の正規化に再利用できる $EC_n(v)$ は $S_n(v)$ が書き換わっていないもの，つまり $n \leq rwd(v)$ である $EC_n(v)$ すべてと $n \geq rwd(v)$ である $EC_n(v)$ のうち $S_n(v)$ が変化していないものである．

異なる $EC_n(v)$ に対して $ECHsh^{(n)}(v)$ の衝突が起こらなかった場合，McKay のグラフ正規化アルゴリズムにおける初回安定細分化の彩色クラスと差分適用グラフ正規化アルゴリズムにおける初回安定細分化の彩色クラスは等しくなる．もし $ECHsh^{(n)}(v)$ に衝突が起きた場合，例えば $ECHsh^{(n)}(v)$ と $ECHsh^{(n)}(w)$ が衝突した場合でも，$ECHsh^{(n+1)}(v)$ と $ECHsh^{(n+1)}(w)$ となりうるため，安定細分化の結果が離散的にならないとは限らない．

Require: 書換え前彩色グラフ (X_1, φ_1)，書換え後彩色グラフ (X_2, φ_2)，書換え前の初回安定細分化有限トライ木 *trie*
Ensure: 書換え後の初回安定細分化有限トライ木 *trie*
  for $v \in \mathcal{V}(X_1) \setminus \mathcal{V}(X_2)$ **do**
    $trie.delete(v)$
  end for
  $trie.root.Vertices := getNewVertices()$
  $halfwayVertices := trie.root.Vertices$
  $nextHalfwayVertices := \{\}$
  $i := -1$
  repeat
    $i++$
    $reverseStack := getRewriteDistanceVertices(i)$
    $goBack(reverseStack)$
    $goAhead(halfwayVertices)$
  until $\neg isDiscrete(i-1)$ *and* $isRefined(i)$
  if $\neg isDiscrete(i-1)$ *and* $\neg isRefined(i)$ **then**
    $collectExtraNodes(trie, i)$
  end if
  return *trie*

図 3.1　差分適用初回安定細分化アルゴリズム （文献 [12] p26 より引用）

3.3.2　差分適用グラフ正規化アルゴリズム概要

書換え前のグラフの $EC_n(v)$ のハッシュ値をトライ木内に保持して再利用しつつ，書換え後のグラフの初回安定細分化を行うためのアルゴリズムを図 3.1 に示す．ただし，入力と出力が逆であったため，本論文への掲載時に修正している．また，図 3.1 のうち，$goBack(reverseStack)$ の処理を図 3.2 に，$goAhead(halfwayVertices)$ の処理を図 3.3 に，$collectExtraNodes(trie, i)$ の処理を図 3.4 に別掲する．なお，引用上の補足として，前述したとおり $ECHsh^{(n)}(v)$ を彩色としている関係上, 図 3.3 における *color* は $ECHsh^{(n)}(v)$ であり，$getColor(v, i)$ は $ECHsh^{(n)}(v)$ を算出する処理に他ならない．

トライ木の情報の更新

この初回安定細分化アルゴリズムでは，書換え前のグラフの EC_n を保持しているトライ木を初期状態とし，$i = 0$ から順にトライ木内の EC_i を書換え後のグラフのものに更新

Require: 後退対象のグラフ頂点スタック $reverseStack$
Ensure: 後退対象のグラフ頂点スタック $reverseStack$
  while $notEmpty(reverseStack)$ **do**
    $v := reverseStack.pop$
    $tmpNode := v.owner$
    $tmpNode.Vertices := tmpNode.Vertices \setminus \{v\}$
    if $tmpNode.depth > i$ **then**
      $halfwayVertices.push(v)$
    end if
    while $tmpNode.depth > i$ **do**
      if $isEmpty(tmpNode.Vertices)$ **then**
        $tmpNode.delete()$
      end if
      $tmpNode := tmpNode.parent$
    end while
    $tmpNode.Vertices.push(v)$
  end while
  return $reverseStack$

図 3.2　ハッシュ値が変化したグラフ頂点の後退処理 $goBack(reverseStack)$(文献 [12] p27 より引用)

Require: 前進対象のグラフ頂点をもつトライ木頂点スタック $halfwayVertices$
Ensure: 前進対象のグラフ頂点をもつトライ木頂点スタック $halfwayVertices$
  while $notEmpty(halfwayVertices)$ **do**
    $v := halfwayVertices.pop$
    if $notLeaf(v.owner)$ *or* $notSingleton(v.owner.Vertices)$ **then**
      $color := getColor(v, i)$
      if $isExistChild(v.owner, color)$ *and* $isLeaf(v.owner.child[color])$ **then**
        $nextHalfwayVertices := nextHalfwayVertices \cup v.owner.child[color].Vertices$
      end if
      $v.owner.child[color].Vertices.push(v)$
      $v.owner.Vertices = v.owner.Vertices \setminus \{v\}$
      $nextHalfwayVertices.push(v)$
    end if
  end while
  $halfwayVertices := nextHalfwayVertices$
  return $halfwayVertices$

図 3.3　未処理のグラフ頂点の前進処理 $goAhead(halfwayVertices)$(文献 [12] p27 より引用)

Require: 有限トライ木 *trie*
Ensure: 有限トライ木 *trie*，整数 i
  for $v \in \mathcal{V}(X_2)$ **do**
    $tmpNode := v.owner$
    $tmpNode.Vertices := tmpNode.Vertices \setminus \{v\}$
    while $tmpNode.depth > i + 1$ **do**
      if $isEmpty(tmpNode.Vertices)$ **then**
        $tmpNode.delete()$
      end if
      $tmpNode := tmpNode.parent$
    end while
    $tmpNode.Vertices.push(v)$
  end for
  return *trie*

図 3.4　離散的な彩色が得られなかった際の終了処理 $collectExtraNodes(trie, i)$(文献 [12] p28 より引用)

する作業を McKay のグラフ正規化アルゴリズムの終了条件を満たすまで繰り返す．その後，終了時の i に対し，$n > i$ である EC_n を削除することで，書換え後のグラフ構造に対するトライ木が保持する EC の正当性を保証している．

トライ木を更新する過程において, 書換えによってグラフ頂点 v の深さ m の n-近傍グラフ $S_m(v)$ が書換えられた結果，v の彩色が少なくとも深さ $i = m$ の時点で $ECHsh^{(m)}(v)$ であることががわかっている状態を，$ECHsh^{(m)}(v)$ をラベルにもつノード N に v を格納することで表現している．すなわち，彩色が深さ $i = m$ の時点で $ECHsh^{(m)}(v)$ である頂点 v は，ノード N またはその子孫ノードに格納される．なお，更新が完了した時点で葉ノードには書換え後のグラフの各頂点のみが格納されている．また，トライ木の処理中，各トライ木のノードには 2 つ以上のグラフ頂点が格納されることもある．

図 3.1 にも示されているとおり，初回安定細分化アルゴリズムではまずグラフ書換え後の全頂点がトライ木内のどこかのノードに格納されている状態にする．これを $v \in \mathcal{V}(X_1) \setminus \mathcal{V}(X_2)$ である頂点，すなわち書換えによって削除された頂点 v をトライ木内から削除し，$v \in \mathcal{V}(X_2) \setminus \mathcal{V}(X_1)$ である頂点，すなわち書換えによって新たに追加された頂点 v をトライ木の根ノードに格納することによって実現する．

その後，i を 0 から 1 ずつ増加させつつ，以下の動作を繰り返す．

(i) EC_i が更新された可能性のある頂点の収集

$rwd(v) = i$ であるグラフ頂点 v (ただし $v \in \mathcal{V}(X_1) \cap \mathcal{V}(X_2)$) は書換え時に $EC_i(v)$ が変化した，すなわち $ECHsh^{(i)}(v)$ が変化した可能性がある．この条件を満たす頂点を (ii) の後退処理で移動させるため，後退対象のグラフ頂点スタック *reverseStack* に登録する．

(ii) 後退処理 (*goBack*(*reverseStack*))
具体的な処理を図 3.2 に記載する．
(i) で *reverseStack* に登録した頂点 v を，各 v の葉ノードから深さ i の祖先ノードへ移動する．ただし，v の格納されている深さが i 以下であった場合は移動しない．v の移動によりトライ木のノード N 自身および N の子孫ノードに頂点が 1 つも格納されていない状態になった場合，N を削除する．
(ii) で移動した v は (iii) の前進処理の対象にもなるので，前進対象のグラフ頂点を持つトライ木頂点スタック *halfwayVertices* に登録する．

(iii) 前進処理 (*goAhead*(*halfwayVertices*))
具体的な処理を図 3.3 に記載する．
halfwayVertices に登録された各頂点が格納されているノードを調べる．このとき，頂点 v が一元彩色クラスに属しているならば v のみがトライ木の葉ノードに格納されている．この状態の場合は葉ノードに格納されたままにする．逆に，v が一元彩色クラスに格納されていない場合，$ECHsh^{(i)}(v)$ を計算し，$ECHsh^{(i)}(v)$ をラベルとする子ノードへ移動させる．

(iv) 終了条件判定
(i)〜(iii) の繰り返しは，グラフ内の全頂点に相異なる $ECHsh_i(v)$ が与えられている，すなわち彩色が離散的となった場合か，あるいは細分化した際に $ECHsh_i(v)$ の種類が増えなかった，すなわち安定細分化が得られた時に終了する．当アルゴリズムでは，前者を *isDeicrete*(i) で，後者を *isRefined*(i) で判定している．
isDeicrete(i) は彩色が離散的となったかどうか，すなわち，グラフの全頂点がトライ木の深さ $i+1$ 以下の相異なる葉ノードに格納されているかどうかを判定する．このとき，グラフ頂点を格納するのは深さ $i+1$ のトライ木ノードか葉ノードのみであること，および後退処理・前進処理の内容から深さ $i+1$ 未満にノードには複数のグラフ頂点を格納されないことから，*isDeicrete*(i) は $i+1$ がトライ木の高さと一致し，なおかつ深さ $i+1$ のトライ木の葉ノードにはグラフ頂点が 2 つ以上格納されていないことを判定すれば十分であることがわかる．トライ木全体の高さを管理するために，トライ木本体とは別に，トライ木のそれぞれの深さにグラフの各頂点がいくつ格納されているかを管理するための配列 *Distribution* と，*Distribution*

のうち個数が 0 でない最大のインデックス max_index を保持し，トライ木内の各頂点が移動する度に更新することで，$isDeicrete(i)$ の判定に利用している．
$isRefined(i)$ は細分化の際に $ECHsh_i(v)$ の種類が増加しているかどうかを判定している．$isRefined(i)$ の判定のために，$ECHsh_{i-1}$ から $ECHsh_i$ でハッシュ値の種類がいくつ増えたかを保持する配列 $Increase$ を用意し，トライ木の枝が追加・削除される度に更新している．これはハッシュ値の衝突がない限り EC_{i-1} から EC_i で有界拡張隣接関係の種類がいくつ増えたかと同義であり，$isRefined(i)$ は $Increase[i]$ が 0 かどうかで判定可能である．

細分化終了後の正当性の維持

前節にて $isRefined(i)$ が偽となって繰り返しが終了した場合，書換え後のグラフの初回安定細分化は離散的でないまま終了している．このとき $i+1$ より大きい深さのノードの情報はグラフ書換え後への更新が行われておらず，深さ $i+1$ のノードへの回収を行う処理が必要である．その処理を行うのが図 3.4 にも示した $collectExtraNodes(trie, i)$ である．これはグラフの各頂点が格納されている深さを調べ，深さ n が $n > i+1$ であるならば深さ $i+1$ まで回収する，という処理をグラフの全頂点に対して行う処理である．

3.4　差分適用グラフ同型性判定

この節では，文献 [12] で提案された差分適用グラフ同型性判定アルゴリズムのうち，差分適用グラフ正規化アルゴリズム以外の点について述べる．3.4.1 節，3.4.2 節，3.4.3 節は文献 [12] に基づく．

3.4.1　正規ラベルの保持

3.3 節で述べたグラフ正規化を行うことで，グラフの各頂点にはハッシュ値が割り振られる．ただし複数のグラフ頂点の隣接情報が全く同じ場合や異なる複数の彩色クラスに属するグラフ頂点に同一のハッシュ値が与えられてしまった場合は各頂点のハッシュ値はその頂点と一対一で対応するものではないため，そのままでは正規化を行なった結果としての正規なラベルとしては使用できない．

そこで，各グラフ頂点に対して

(頂点のハッシュ値, 同じハッシュ値内での通し番号)

の組を与え，各グラフ頂点と一対一で対応する正規ラベルとする．ハッシュ値が衝突した際の一意な通し番号の割り振りに関してはトライ木上の位置，および離散的な初回安定細分化が得られなかった場合に生成する探索木によって得られる順序に従う．

なお，上記の正規ラベルの取得は，異なる彩色クラスに属するグラフ頂点に異なるハッシュ値が与えられる限り，細分化終了後に離散的な初回安定細分化が得られなかった際に行う正当性の処理内とその後の探索木の生成処理内で完結させることが可能であるため，正規ラベルの取得の時間計算量は差分適用グラフ正規化アルゴリズムの時間計算量以下となる．

3.4.2　正規化グラフの差分表現

まず，離散的な彩色をもつグラフの一意表現について考える．ここで，グラフの各頂点 v に対する

- 頂点 V の彩色
- 頂点 v の隣接頂点の彩色の多重集合

の組を頂点 v の隣接情報とし，全頂点の隣接情報を集合としたものを考える．これは，任意のグラフ構造に対して一意な彩色が得られる，すなわち異なるグラフ構造であれば少なくとも 1 点以上は異なる彩色となることを考えると，全てのグラフ構造に対して一意な表現であるといえる．以降，全頂点の隣接情報の集合を (グラフに対する) グラフの一意表現とする．

モデル検査を行う際，生成したグラフの一意表現をそのまま格納するとグラフ内の辺に応じた時間計算量となってしまい，非常に効率が悪い．そこで，時間計算量改善のために，グラフ同型性判定時に必要になった際に改めて処理できるような形でグラフの一意表現の差分情報を格納する．そこで，対称差演算は交換法則と結合法則を満たすことから演算も容易であり，

$$A \triangle C = (A \triangle B) \triangle (B \triangle C)$$

といった形で差分の合成も容易である対称差を，書換え前後のグラフの一意表現の差分情報として用いる．これによって，ある状態の具体的なグラフの一意表現が必要になった際も，”初期状態のグラフの一意表現と，初期状態からある状態までの各遷移に登録された対称差全て”の対称差をとるだけでグラフの一意表現が生成できる．また，別の状態と同一の状態であるか比較する際は，あらかじめ何らかの形で状態空間に関する有向全域木を持っておき，書換え前後の差分情報を初期値として，状態空間に関する有向全域木と書換

えによって作られる遷移の候補からなる基本閉路を辿りながら，路に対応する 2 つのグラフの対称差が空であるかを判定するだけでよい．なお，全域木と全域木に属する 2 頂点間の有向辺によって作られる基本閉路はただ 1 つであるので，状態空間内の路の探索を行わずに済むということも，状態空間に関する全域木を持っておくことの利点である．

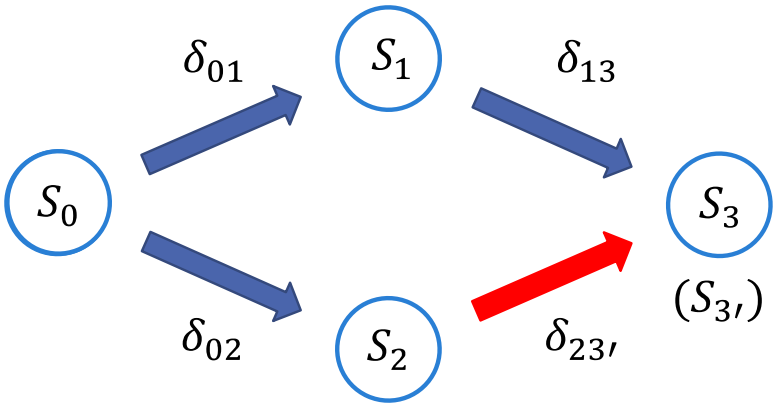

図 3.5　遷移の一例

例えば，図 3.5 のように状態空間内に状態 S_1 から S_3 があり，$S_i \cdot S_j$ 間の一意表現の対称差が δ_{ij} であるとき，S_3 と同一状態の候補である S'_3 が S_3 と同一であるかを判定するためには，S_2 と S'_3 の対称差を $\delta_{23'}$ として

$$\delta_{23'} \triangle \delta_{02} \triangle \delta_{01} \triangle \delta_{13}$$

が空集合であるかを判定すればよい．

なお，何らかの形で持つ状態空間に関する有向全域木に関しては，4 章にて詳細を述べる．

3.4.3　差分を適用したハッシュ値の生成

状態に対するハッシュ値を生成することで要素どうしの比較処理をハッシュ値の比較に置き換えることが可能である．ただし，状態に対するハッシュ値を算出する処理の時間計算量が悪ければモデル検査全体の時間計算量が悪くなる可能性があるため，少なくともグラフ正規化時に時間計算量が劣らないハッシュ値の算出手法を実現したい．

ここで，グラフの一意表現は隣接情報の集合であることから，集合に対するハッシュ関数を考える．ただし，状態内の各隣接情報のハッシュ値は既に計算済みであるものとする．

文献 [3] で提案された，グラフを対象にしたハッシュ関数は以下のとおりである．ただし，x はグラフ構造の各頂点，S は状態，すなわちグラフの頂点集合と辺集合の組をさす．

$$hash(S) = \left(\sum_{x \in S} hash(x)\right) \bigoplus \left(\prod_{x \in S} hash(x)\right)$$

このハッシュ関数は一般的な集合のハッシュ値としても有効なものであり，差分適用同型性判定アルゴリズムでも集合の要素のハッシュ値の総和と相乗の各ビットごとの排他的論理和をハッシュ値として用いる．ただし，書換え前後のグラフの一意表現の差分は一部のみであるから，書換え前のグラフの一意表現のハッシュ値に対して書換え前後のグラフの一意表現の差分に応じた計算を行うのみで書換え後のグラフのハッシュ値を生成したい．そこで，ハッシュ値の計算が 2^N を法とする剰余類環 $(\mathbb{Z}/2^N\mathbb{Z})$ であることに着目する．このとき，総和に関しては整数の加減算が $(\mathbb{Z}/2^N\mathbb{Z})$ 上での加減算に一致するため，書換え前後で追加される隣接情報のハッシュ値を加算し，書換え前後で削除される隣接情報のハッシュ値を減算する処理のみでも正当性が維持される．一方，相乗に関しては整数の乗除算が $(\mathbb{Z}/2^N\mathbb{Z})$ 上での乗除算に一致しないため，書換え前後で追加される隣接情報のハッシュ値を乗算し，書換え前後で削除される隣接情報のハッシュ値を除算する処理では正当性が維持されない．

上記の状況を改善するため，文献 [12] において提案されているのが以下のハッシュ関数である．

$$hash(S) = \left(\sum_{x \in S} hash(x)\right) \bigoplus \left(\prod_{x \in S} (2hash(x) + 1)\right)$$

総和の部分に関しては前述のとおり更新の際の正当性が維持されている．更に相乗に関して，各隣接情報のハッシュ値に 2 を掛けて 1 を足す，すなわち相乗の全ての要素を奇数にすることで，書換え前後で追加される隣接情報のハッシュ値に 2 を掛けて 1 足したものを乗算し，書換え前後で削除される隣接情報のハッシュ値に 2 を掛けて 1 足したものを除算する処理のみで正当性が維持されるようになっている．これは奇数と 2^N が互いに素であることから任意の奇数 x に対して $xy = 1 \pmod 2^N$ となる y が存在する，すなわち任意の奇数 x に乗法逆元が存在するためである．

3.4.4　グラフ頂点の彩色情報変化の検知

3.4.2 節で述べたような書換え前のグラフ構造の表現から書換え後のグラフ構造の表現および正規化グラフの差分情報を生成する際，書換え前後のグラフの正規化の結果の両方

がともに必要である．しかし，差分を適用したグラフ書換え後の正規化結果と比較する際，コピーして保持しておいた書換え前のトライ木の頂点を逐一走査するという手法は非常に効率が悪い．そこでこの節では，差分を適用した書換え後のグラフの正規化中に，頂点のハッシュ値の更新が起こったタイミングでその頂点のハッシュ値を配列に格納することで，書換え前後の各頂点のラベルの変化を検知する．

まず，書換え前のグラフ内の各頂点にあらかじめ 0 から始まる適当な通し番号を振っておく．次に，書換え前の頂点の数と同じ数の正規ラベルを格納する配列を確保し (今後この配列をラベル配列と呼ぶ)，その各要素に正規ラベルが持ちえない形を代入して初期化する．たとえば，正規ラベルの通し番号を 0 から割り振るのであれば，ラベル配列内の各要素に (0,-1) を代入しておく．

その後，書換え前のグラフの正規化処理を行う．このとき，トライ木内で管理されているグラフの各頂点のハッシュ値が更新される

- 書換え時に別の頂点が削除されたことによりトライ木内をさかのぼった頂点
- 後退処理の対象に選ばれ，トライ木内をさかのぼる処理
- 後退処理を行なった際に不要になったノードに所属していたことによる，ノードの削除と同時にトライ木内をさかのぼる処理
- 前進処理を行ない，対応する子ノードに渡される処理
- 離散的な彩色が得られなかった際の終了処理でトライ木内をさかのぼる処理

以上 4 つの処理を行う際，ラベルが更新された頂点の通し番号をインデックスとして，

(新しいハッシュ値, 正規ラベルの通し番号の最初の数字)

の組をその頂点の正規ラベルとしてラベル配列に格納する．この処理により細分化が進行するとともにラベル配列内の正規ラベルが更新されていく．

最後に，得られた初回安定細分化が離散的でなかった場合，3.4.1 節に述べた順序にしたがって通し番号を振り，ラベル配列のそれぞれの要素に格納する．

書換え前の全ての頂点の正規ラベルが得られたら，書換え後の正規ラベルを得るべく書換え前と同様の手続きを行っていく．ただし通し番号に関しては，新しく追加する頂点に対しては書換え前のグラフの最後の番号の次から割り振り，書換え前後の両方に存在する頂点に関しては通し番号は書換え前と同じものを割り振る．つまり，書換えにより消滅する頂点の通し番号は書換え後のグラフでは欠番となる．また，書換え後のラベル配列を確保する際も (振った通し番号のうち最も大きいもの $+1$) の大きさの配列を確保する．

その後，書換え前のグラフの正規化結果に差分を適用して書換え後のグラフの正規化を

行うと同時に，書換え前グラフで行なったものと同じ処理を行って書換え後のラベル配列に格納していく．このとき書換え後のラベル配列に格納されるのは新規に追加した頂点を含むラベルが更新された頂点の新しい正規ラベルのみであるため，ある頂点の正規ラベルが更新されたかは書換え後のラベル配列のその頂点と対応する要素の値が正規ラベルとして持ちえる値かどうかを調べるだけで確認できる．

3.4.5　隣接頂点の彩色情報変化の検知

3.4.4 節では書換え前後のグラフ頂点のうち，どの頂点に正規ラベルの変化があったかを検知する方法を述べた．しかしグラフ構造の表現の一部である各頂点の隣接表現は各頂点自身のラベルが更新された時だけでなく，書換え時に隣接した頂点のみが更新された際も更新する必要がある．

したがって，差分適用正規化時にラベル配列に格納する対象に

- 後退処理の対象に選ばれたが，実際にノードをさかのぼらなかった頂点
- 書換え時に別の頂点が削除されたことによりノードをさかのぼった頂点の隣接頂点

を追加する．

具体的には，前者は図 3.2 において *reverseStack* に入っており v に代入されたものの，$tmpNode.depth = i$ であったが故に **if** $tmpNode.depth > i$ **then** から **end while** までの処理の対象にならず，同じ *tmpNode* に戻された頂点，すなわち隣接する頂点が後退処理の対象に選ばれた頂点である．また，後者は一切の処理の対象にならないため，頂点ごとに個別に対応する必要がある．ただし，これらの頂点に関しては，正規ラベルが取りえない形でかつラベル配列の初期化に使用しなかった値を格納してもよい．例えば，初期化時に (0,-1) をラベル配列の各要素に代入したのであれば，フラグとして (0,-2) を格納してもよい．

これらを正規ラベルの更新が起こった頂点とともに書換え後のラベル配列に格納することによって，書換え後のラベル配列に格納された正規ラベルは書換え後のグラフの正規化結果の他に該当する通し番号の頂点の隣接情報が更新されている可能性があるかどうかのフラグの意味ももつようになる．

3.4.6　書換え後の一意表現と書換え前後の差分情報の生成

書換え後のグラフの正規化が完了した後，差分情報としての書換え前後のラベル配列，および書換え後のグラフ，書換え時に削除した頂点の一覧を利用することで正規化グラフの書換え前後の差分情報と書換え後のグラフの一意表現を得られる．アルゴリズムを図 4.14 に示す．ただし，書換え前のラベル配列 *beforeLabels* には書換え前の全ての頂点のラベルが，書換え後のラベル配列 *afterLabels* には 3.4.5 節で述べた頂点のラベルのみが格納されているものとする．

書換え時に削除した頂点のグラフ情報の削除

書換え後のラベル配列 *afterLabels* 内において正規化中に更新がなかった要素のインデックスを通し番号を持つ頂点は

- 書換え時に削除された頂点
- 書換え後のグラフ正規化時に更新されていない頂点

のいずれかであるが，*afterLabels* からそのどちらかを判断することはできない．

そこで，書換え時に削除した頂点の一覧を用いて削除した頂点の隣接情報 *info* を削除することで，書換え後のグラフ *afterGraph* に存在する頂点の書換え前の隣接情報のみをグラフの一意表現 *graphExpr* に残すことができる．

削除した隣接情報 *info* は差分情報 *diffInfo* に追加する．

各頂点の書換え前後の隣接表現の追加・削除

書換え時に削除した頂点のグラフ情報の削除が完了したら，*afterLabels* に正規ラベルが格納されている各頂点，すなわち書換え前後で隣接表現が変化した可能性のある頂点や書換えによって追加された頂点の

- 書換え前の隣接情報の削除
- 書換え後の隣接情報の追加

をグラフの一意表現 *graphExpr* に対して行い，同時に必要に応じて差分情報 *diffInfo* を更新していく．

Require: 書換え後のグラフの一意表現 *graphExpr*, 書換え前後の差分情報 *diffInfo*
Ensure: 書換え前のグラフの一意表現 *graphExpr*, 書換え時に削除した頂点の一覧 *deletedVertices*, 書換え前のラベル配列 *beforeLabels*, 書換え後のラベル配列 *afterLabels*, 書換え後のグラフ *afterGraph*

```
for v ∈ deletedVertices do
    info := graphExpr.search(v)
    graphExpr.delete(info)
    diffInfo.add(info)
end for
for l ∈ afterLabels do
    v := l.owner
    beforeInfo := graphExpr.search(v)
    if isExist(beforeInfo) and beforeInfo ∉ diffInfo then
        diffInfo.add(beforeInfo)
        if beforeInfo ∈ overlapExpr then
            overlapExpr.delete(beforeInfo)
        else
            graphExpr.delete(beforeInfo)
        end if
    end if
    afterInfo := collectinfo(v, beforeLabels, afterLabels)
    if afterInfo ∈ graphExpr then
        overlap.add(afterInfo)
    else
        graphExpr.add(afterInfo)
        if afterInfo ∈ diffInfo then
            diffInfo.delete(afterInfo)
        else
            diffInfo.add(afterInfo)
        end if
    end if
end for
return graphExpr, diffInfo
```

図 3.6　差分を適用した書換え後のグラフの一意表現と差分表現の生成アルゴリズム

書換え前の隣接表現の削除

まず，頂点 v の書換え前の隣接情報を削除する．

最初にグラフの一意表現 *graphExpr* の中から頂点 v の書換え前の隣接情報を探し，*beforeInfo* とする．頂点 v が書換え後に追加された頂点の場合は *beforeInfo* は存在しないので，これ以降の処理は行わない．

beforeInfo が存在した場合，次に *beforeInfo* がグラフの一意表現 *graphExpr* に重複されて追加された隣接表現の集合 *overlapExpr* に含まれていないかを確認する．もし含まれていた場合は *overlapExpr* から *beforeInfo* を削除して処理を終了する．この処理はグラフの一意表現 *graphExpr* から *beforeInfo* を削除する処理と同義だが，グラフの一意表現には同一の隣接情報が 2 つ含まれることが許されないため，後述の追加処理で隣接情報の重複が起こった場合に一時的に *overlapExpr* に格納していることから起こる処理である．含まれていない場合はグラフの一意表現 *graphExpr* から *beforeInfo* を削除し，処理を終了する．

書換え後の隣接表現の追加

次に，頂点 v の書換え後の隣接情報を追加する．

まず，v の書換え後の隣接情報 *afterInfo* を取得する．

これを行う処理 $collectinfo(v, beforeLabels, afterLabels)$ の内容は以下のとおりである．

1. 書換え後のグラフ *afterGraph* から各隣接頂点の通し番号を取得
2. v および各隣接頂点の通し番号をインデックスとして書換え後のラベル配列 *afterLabels* の要素を確認
3. 正規ラベルが格納されていればその要素を *afterInfo* に格納，正規ラベルが格納されていなければ書換え前のラベル配列 *beforeLabels* の同じインデックスの要素を *afterInfo* に格納

書換え後の隣接情報 *afterInfo* を取得したら，次にグラフの一意表現 *graphExpr* に *afterInfo* が含まれているかを確認する．もし含まれている場合，*graphExpr* には同一の隣接情報が 2 つ含まれることが許されないので，重複されて追加された隣接表現の集合 *overlapExpr* に追加する．

グラフの一意表現 *graphExpr* に *afterInfo* が含まれていなかった場合，*graphExpr* に *afterInfo* を追加し，その後差分情報 *diffInfo* に *afterInfo* が含まれているか確認する．もし含まれている場合，*afterInfo* は書換え前の隣接情報の削除処理においても削除された情

報である．すなわち，*afterInfo* は 1 回の書換えで削除と追加が同時に行われた情報であるので，*diffInfo* から *afterInfo* を削除して処理を終了する．*diffInfo* に *afterInfo* が含まれていない場合，当然ながら *afterInfo* は新しく追加された隣接情報であるので，*diffInfo* に *afterInfo* を追加して処理を終了する．

以上のアルゴリズムにより得られた正規化グラフの書換え前後の差分情報 *diffInfo* と書換え後のグラフの一意表現 *graphExpr* は 3.4.4 節で各頂点に振った通し番号に関係なく一定であるので，3.4.4 節における通し番号の割り振り方は任意である．また，もし 3.4.3 節に示されたグラフ構造のハッシュ値を計算する場合は，図 4.14 内の *beforeInfo* への情報の追加または *beforeInfo* からの削除を行うと同時に逐次ハッシュ値を更新すればよい．

第 4 章

基本閉路長削減手法

この章では，本研究にて提案する手法である基本閉路長削減手法に関して述べる．基本閉路長削減手法が差分適用グラフ同型性判定手法のどこに位置するのかに言及したのち，提案する基本閉路長削減手法について具体的に述べる．

4.1　基本閉路長削減手法の適用箇所

3.1 節で紹介した，差分適用グラフ同型性判定アルゴリズムの 4 ステップをここに再掲する．

(i) 書換え前のグラフ，書換え前のグラフの正規化結果，および書換え後のグラフを利用して，書き換え後の正規化結果を得る．
(ii) 書換え前のグラフの一意表現，書換え前のグラフの正規化結果と書換え後のグラフの正規化結果の差分から，書換え後のグラフの一意表現と書換え前後の一意表現の対称差を得る．
(iii) 書換え後のグラフの一意表現を用いて，状態空間内の同一状態候補を検索する．
(iv) 候補が存在すれば，書換え前後の一意表現の対称差，および状態空間内の各状態の一意表現の対称差を用いて，書換え後のグラフと同一状態候補の一意表現の対称差を計算する．

基本閉路長削減手法は，上記の (iv) のステップにおいて使用する一意表現の対称差の量の削減を行うためのアルゴリズムである．

なお，3.4.2 節にて上記の各ステップにおける “一意表現の対称差”を差分情報と呼ぶことに触れたため，この章以降では今後差分情報という単語で対称差を扱う．

4.2　状態空間の基本有向全域木

基本閉路長削減手法に関する前提として，状態空間の有向全域木の最も基本的な取得方法を述べる．

状態空間の各状態に対して，その状態が新しく状態空間に登録された際に同時に登録された遷移は，初期状態を除いてただ 1 つ存在する．これを生成遷移と呼ぶ．状態空間内の生成遷移のみを抽出することで，初期状態から各状態への路が一意に定まり，かつ閉路をもたない構造，すなわち状態空間の全状態を対象にした有向木を抽出できる．本論文では今後，状態空間内の生成遷移からなる有向全域木を状態空間の基本有向全域木と呼ぶ．

一例として，図 4.1 にある状態空間，図 4.2 に図 4.1 の状態空間の基本有向全域木を示す．ただし，各状態の番号は深さ優先で状態空間を構築した際，その状態から別の状態への展開を試みた順番であり，図 4.2 における実線の遷移が生成遷移，点線が非生成遷移である．図 4.2 から，基本有向全域木内に初期状態から初期状態以外の全状態への路がただ 1 つ存在していることを確認できる．

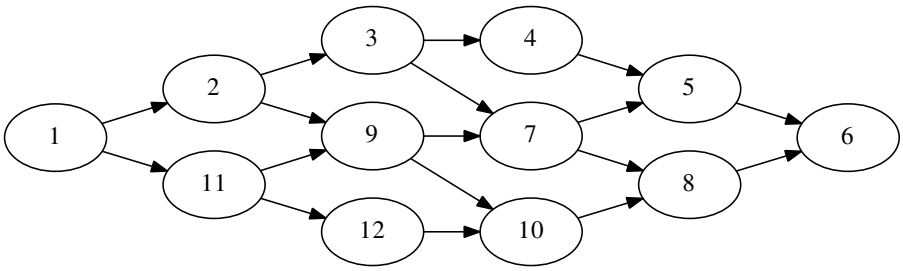

図 4.1　展開中の状態空間の一例

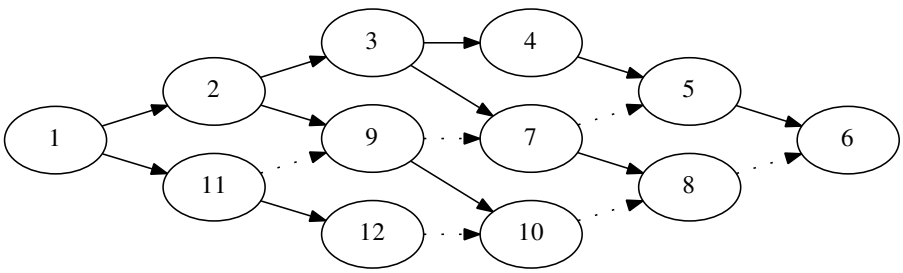

図 4.2　図 4.1 の状態空間の基本有向全域木[*2]

4.3　基本閉路長削減手法における制約

差分適用グラフ同型性判定アルゴリズムのステップ (iv) においては 3.4.2 節で述べた通り，状態空間の有向全域木と書換えによって作られる遷移の候補からなる基本閉路を辿る．この辿るという操作において有向全域木の各遷移の情報を逐次的に収集する必要があるため，事前の計算なしに複数の遷移を遡ることができない．更に，状態空間というグラフ構造自体が状態空間の構築中に常に更新される．そのため，事前に有向木に関する情報の計算を行うということができず，また，状態空間の更新の度に全ての状態に関する情報を計算し直すのは，状態空間の展開時の時間計算量を悪化させる原因になりかねない．

以上から，書換え後のグラフと同一状態候補の一意表現の対称差を計算する処理の処理量を改善するにあたっては，同処理の 1 回毎に可能な限り少ないステップで有向全域木の各遷移の情報を逐次的に集めた後，可能な限り小さい処理の実行によって差分情報の使用量を削減することが求められている．そこで，基本閉路長削減手法では，状態空間の有向全域木を，有向全域木であることを維持しながら操作することで，同処理の 1 回毎に処理する差分情報の量を改善する．

4.4　基本閉路長を削減する手法の従来案

4.3 節で挙げた，有向全域木であることを維持しながら，状態空間の有向全域木を操作することによって算出時に使用する差分情報の量を改善する手法は，[12] にて提案されている．本節では，その手法および手法の限界について述べる．本節の手法に関しては [12] に基づく．

「深さ優先探索による状態の展開において，ある状態から他の状態への路の長さはその状態の祖先につながる遷移を多くもつ木構造の方が小さくなる[*3]」と仮定する．この仮定に基づいて，状態空間内に新しく生成遷移以外の遷移が登録された際に，状態空間の有向全域木における書換え後の状態への遷移を以前使用していた遷移から新しく登録された遷移へ更新することで，状態空間の基本有向全域木より少ない量の差分情報で書換え後のグラフと同一状態候補の差分情報が計算できる．ただし，状態空間の有向全域木が有向全域木であることを維持するため，新しく登録された遷移への更新は，以前使用していた遷移が “遷移を登録する前の状態空間の有向全域木と新しく登録された遷移がなす基本閉路”

[*2] 実線が基本有向全域木に採用されている遷移，破線は採用されていない遷移．

[*3] 文献 [12] pp. 68-69 より引用．

に含まれていた場合に限る．

この手法を図 4.1 の状態空間に適用することで適用例を示す．ただし，図 4.3，図 4.4，図 4.5，図 4.6 において，各状態の番号は図 4.1 と同一の番号を採用している．すなわち，各状態が状態空間に登録された順番でないことをあらかじめ断っておく．

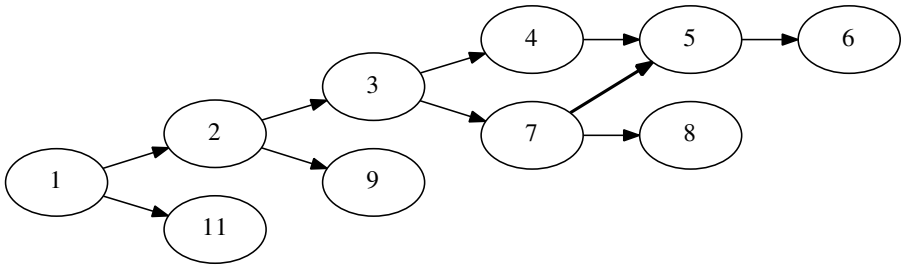

図 4.3　従来の手法における図 4.1 の状態空間の有向全域木の途中状態[*5]

図 4.1 の状態空間を構築する際，最初に出現する非生成遷移は状態 7 から状態 5 への遷移である (図 4.4)．ここで，書換え後の状態である状態 5 への遷移は状態 4 から状態 5 への遷移であるが，この遷移は実線によって作られる状態空間の有向全域木と新しく登録される状態 7 から状態 5 への遷移がなす，

- 状態 3 から状態 4 への遷移と状態 4 から状態 5 への遷移からなる，
  状態 3 から状態 5 への有向路
- 状態 3 から状態 7 への遷移と状態 7 から状態 5 への遷移からなる，
  状態 3 から状態 5 への有向路

の 2 つの路で構成される基本閉路に含まれている．よって，状態空間の有向全域木における状態 5 への遷移を状態 7 から状態 5 への遷移へと更新する (図 4.4)．

この更新によって，状態空間の有向全域木と次に出現する状態 8 から状態 6 への遷移からなる基本閉路の基本閉路長を削減できる．

状態空間の基本有向全域木と状態 8 から状態 6 への遷移からなる基本閉路は，図 4.2 を参照するとわかるとおり

- 状態 3 から状態 4 への遷移，状態 4 から状態 5 への遷移，

[*5] 実線は有向全域木に採用されている遷移，太線は今回新しく登録された遷移．
[*7] 実線は有向全域木に採用されている遷移，点線は有向全域木に採用されていない遷移．
[*9] 実線は有向全域木に採用されている遷移，点線は有向全域木に採用されていない遷移，太線は今回新しく出現した遷移．

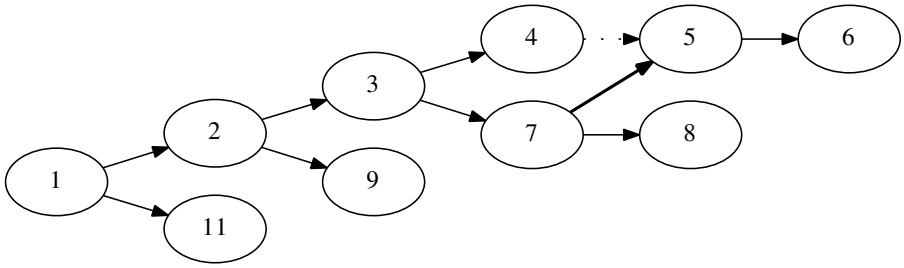

図 4.4　従来の手法における図 4.1 の状態空間の有向全域木の途中状態*7

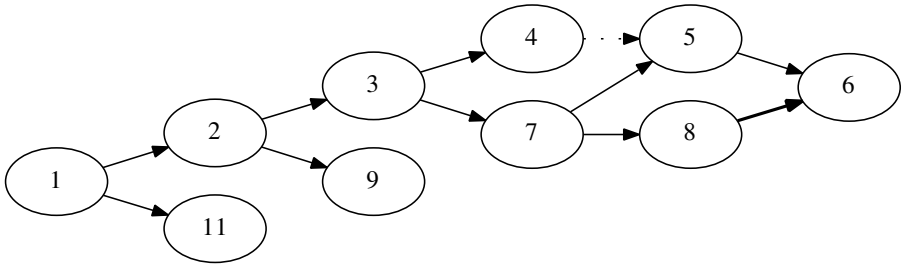

図 4.5　従来の手法における図 4.1 の状態空間の有向全域木の途中状態*9

状態 5 から状態 6 への遷移からなる，状態 3 から状態 6 への有向路

- 状態 3 から状態 7 への遷移，状態 7 から状態 8 への遷移，
  状態 8 から状態 6 への遷移からなる，状態 3 から状態 6 への有向路

であり，基本閉路長は 6 であるが，採用される遷移の更新をおこなった状態空間の有向全域木と状態 8 から状態 6 への遷移からなる基本閉路は，図 4.5 を参照するとわかるとおり

- 状態 7 から状態 5 への遷移と状態 5 から状態 6 への遷移からなる，
  状態 7 から状態 6 への有向路
- 状態 7 から状態 8 への遷移と状態 8 から状態 6 への遷移からなる，
  状態 7 から状態 6 への有向路

であり，基本閉路長は 4 へと削減されている．

更に，書換え後の状態への遷移である状態 5 から状態 6 への遷移は図 4.5 に示された状態空間の有向木と状態 8 から状態 6 への遷移からなる基本閉路に含まれているため，状態 6 への遷移は状態からの遷移から状態 8 からの遷移へと更新される．

このようにして，状態空間の有向全域木に採用される遷移を更新しながら，状態空間を

図 4.1 の状態まで構築した結果を図 4.6 に示す.

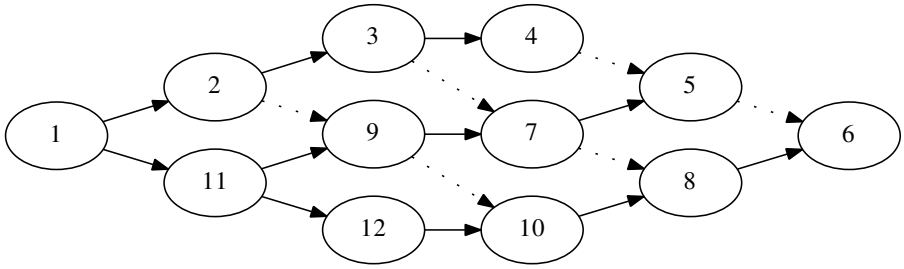

図 4.6　従来の手法における図 4.1 の状態空間の有向全域木[*11]

以上のように，紹介した手法によって状態空間の有向全域木に採用される遷移を更新することで，状態空間と発見された遷移 (およびその候補) がなす基本閉路長の削減が可能である．しかし，先ほど述べた『以前使用していた遷移が “遷移を登録する前の状態空間の有向全域木と新しく登録された遷移がなす基本閉路”に含まれていた場合に限る』という条件付けから，従来の手法によって状態空間の有向全域木に新しく採用される遷移は，状態空間の有向全域木のより浅い状態からより深い状態への遷移に限られる．これにより，有向全域木のより深い状態からより浅い状態へと遡る遷移を多く含む状態空間では遷移の更新回数が小さくなってしまい，その結果計算量の削減割合が小さくなってしまう，ということが考えられる.

例えば，図 4.7 のような状態空間で，状態 6 から他の状態への展開を試みている最中，状態 6 から状態 1 への遷移が状態空間に登録された後に状態 6 から状態 4 への遷移候補が発見された，という状況を考える．ただし，図 4.1 と同様，図 4.7 および図 4.8 の各状態に振られた番号は，深さ優先で状態空間を構築した際の別の状態への展開を試みた順である.

図 4.7 における非生成遷移は状態 4 から状態 1 への遷移・状態 6 から状態 1 への遷移の 2 本がある．しかし，初期状態である状態 1 は有向全域木における根であることから，そもそも有向全域木に状態 1 への遷移が登録されておらず，両遷移は有向全域木へ採用されない．以上，図 4.7 の状況下で，従来の手法によって構築されている状態空間の有向全域木は図 4.8 である．これは図 4.7 の基本有向全域木と変わらない．すなわち，従来の手法では図 4.7 の状態空間においては，基本有向全域木を利用して差分情報を計算する場合と処理量が変わらない.

[*11] 実線は有向全域木に採用されている遷移，点線は有向全域木に採用されていない遷移.

そこで，基本閉路長削減手法では，この従来の手法を参考に，状態遷移を遡る遷移も条件付きで有向全域木に採用することで，より幅広い状態空間に対して計算量を削減する．

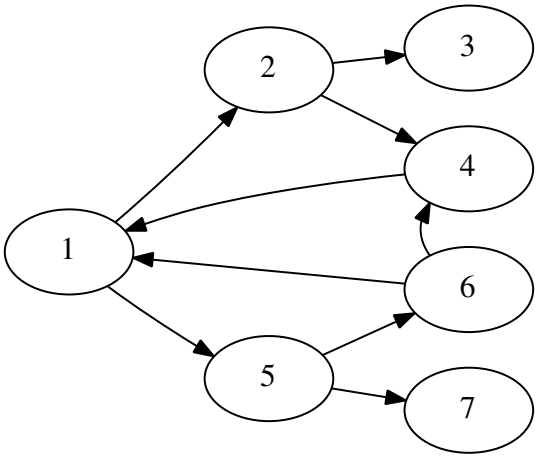

図 4.7　従来の手法では十分に効率化が図れない状態空間の例

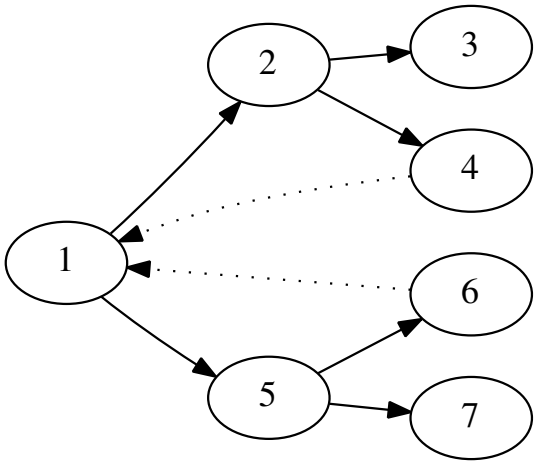

図 4.8　従来の手法による図 4.7 の状態空間の有向全域木[*13]

4.3 節および 4.4 節を踏まえて，これ以降，基本閉路長削減手法の前提と具体的な方法に言及する．

4.5　基本閉路長削減アルゴリズムの前提

基本閉路削減手法は，深さ優先で状態空間を構築する中で

(i) 状態空間の有向全域木の書換え前の状態からの最大距離は，書換え前の状態から距離が近い状態を端点に持つ遷移を多く含む有向全域木ほど小さくなる
(ii) 書換え前の状態から距離が近い状態を端点に持つ遷移は，より直近で状態空間に登録された遷移であることが多い

という 2 点を前提にしている．

前提 (i) に関しては，同一状態数の有向全域木は必ず同一数の有向辺を持っていることから明らかである．これは有向全域木を無向木と捉えなおすとより直感的である．一例として，図 4.8 を無向木と捉えなおした図を図 4.9 に，図 4.8 の有向全域木に状態空間を遡

[*13] 実線は有向全域木に採用されている遷移，点線は有向全域木に採用されていない遷移．

る状態 4 から状態 1 への遷移および状態 6 から状態 1 への遷移を追加し，代わりに状態 2 から状態 4 への遷移および状態 5 から状態 6 への遷移を削除した有向グラフを無向木と捉えなおした図を図 4.10 に示す．距離 0 である状態 6 を端点にとる辺が 1 本から 2 本へ，距離 1 である状態を端点にとる辺が 2 本から 3 本へ増えた結果，図 4.9 の木では 4 であった状態 6 からの最大距離は 3 へと短縮されている．

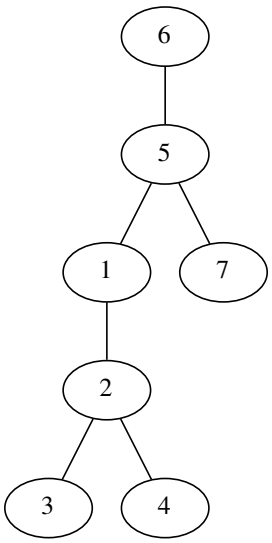

図 4.9　図 4.8 の有向木を無向木と捉え直したときの木

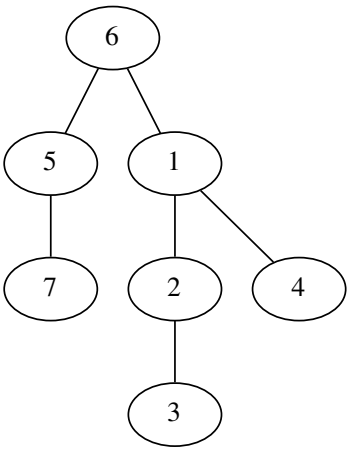

図 4.10　図 4.7 の 2 本の遡る遷移を有向全域木に採用して無向木と捉え直したときの木

前提 (ii) はヒューリスティックな仮定である．例えば，図 4.5 において，書換え前の状態 8 から距離 1 である状態 7 を端点に持つ状態 7 から状態 5 への遷移は，生成遷移である状態 4 から状態 5 への遷移に比べてより直近で状態空間に追加された遷移である．また，図 4.7 において，状態 4 から状態 1 への遷移は状態 2 から状態 4 への遷移に比べてより直近で状態空間に追加された遷移である．

4.6　基本閉路長削減アルゴリズム

4.5 節で述べた 2 点の前提をもとに，基本閉路長削減手法の具体的な手法を述べる．後に再び触れるが，図 4.14 に基本閉路長削減アルゴリズムを示す．

4.6.1　状態遷移を遡る遷移の採用方法

状態遷移を遡る遷移を状態空間の有向全域木に採用することで差分情報の処理量を削減したいが，遡る遷移をそのまま有向全域木に採用してしまうと有向全域木という構造を維持できない．そこで，状態空間の有向全域木内に限って，遷移の方向に関わらず状態空間の基本有向全域木における各状態の深さを基準とし，深さがより浅い状態からより深い状態への遷移として扱う．例えば，図 4.7 の状態空間の構築中，状態 4 から状態 1 への遷移が発見され (図 4.11)，状態空間に登録された際，状態空間の基本有向全域木による状態 1 の深さは 0，状態 4 の深さは 2 であるため，有向全域木には状態 1 から状態 4 への遷移として登録される (図 4.12)．

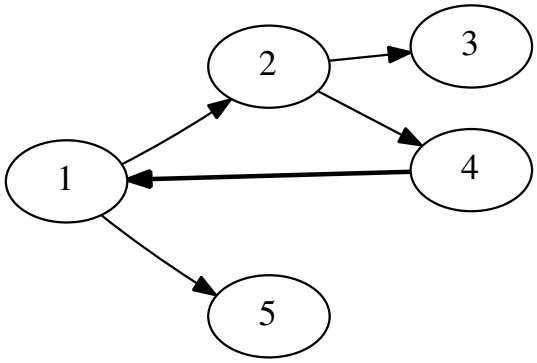

図 4.11　閉路長削減アルゴリズムによる図 4.7 の状態空間の有向全域木の途中状態[*15]

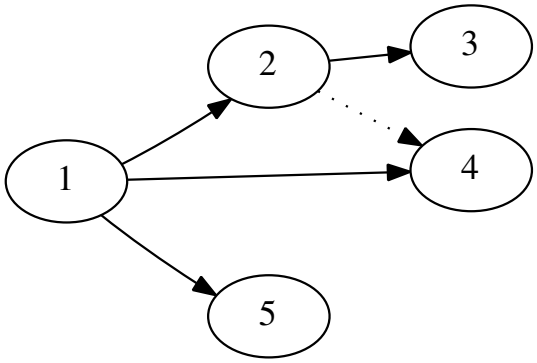

図 4.12　閉路長削減アルゴリズムによる図 4.7 の状態空間の有向全域木の途中状態[*17]

この方法をとることにより，状態空間を遡るような遷移を有向全域木に採用しても，状態空間の有向全域木は初期状態を根とする有向全域木の構造を維持する．

なお，状態空間の基本有向全域木における各状態の深さは，初期状態の深さを 0 とし，各状態が新しく状態空間に登録された際に (書換え前の状態の生成遷移による深さ +1) と与えることで簡単に管理できる．

[*17] 実線は有向全域木に採用されている遷移，太線は新しく発見された遷移．
[*17] 実線は有向全域木に採用されている遷移，点線は採用されていない遷移．

4.6.2　同一の深さの状態間の遷移の採用について

先に述べた通り，状態空間の有向全域木における遷移の方向は状態空間の基本有向全域木の深さを基準とする．ただし，状態空間の基本有向全域木で同一の深さの状態間の遷移は，この基準ではどちら向きの遷移とすればよいか判定できない．更に，図 4.13 のような 3 状態以上の同一の深さからなるループ構造が状態空間内に構成される場合，同一の深さの状態間の遷移を状態空間に採用してしまうと，状態空間の有向全域木内にループが発生する可能性がある．(もちろん，その時点で状態空間の有向全域木は有向全域木の構造を維持していない．)

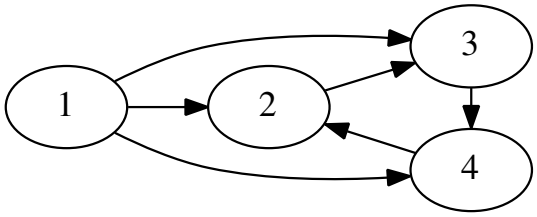

図 4.13　遷移の抽出によってループの発生しうる状態空間の一例

以上のような事態をあらかじめ回避するため，状態空間の有向全域木には同一の深さの状態間の遷移を採用しないことにする．

4.6.3　状態空間の有向全域木と遷移候補がなす基本閉路の辿り方

4.6.1 節，4.6.2 節の採用方針によって，状態空間の有向全域木が有向全域木であることを維持したまま，状態空間により最新の遷移を採用できる．また，状態空間の有向全域木と遷移候補がなす基本閉路は，書換え前の状態と書換え後の状態のそれぞれからお互いの基本有向全域木における深さを比較し，深い方の状態のみ (深さが等しい場合は両方) がその書換え前の状態へと遡ることを繰り返すのみで辿ることが可能である．

ただし，4.6.1 節によって採用された遷移によって遡った場合は，書換え前後の基本有向全域木における深さの差が 1 とは限らない (0 であることはない)．なので，有向全域木の遡りを行う度にお互いの基本有向全域木における深さを比較する必要がある．

4.6.1 節から 4.6.3 節の方針をまとめたものが図 4.14 である．ただし，図 4.14 において，状態空間の有向全域木 $spTree$ は各頂点が状態空間の各状態と相互に対応している

ものの，状態空間とは別の構造として存在している，という形を想定している．また，$V_i.parent$ は状態空間の有向全域木における V_i の書換え元の状態を指す．

前半の **while** 部では，4.6.3 節に則りつつ，状態空間の有向全域木と遷移候補からなる基本閉路を差分情報の対称差を取りながら遡る．図 4.14 上では $V_i.state$ と $V_i.parent.state$ の対称差を $V_i.state \triangle V_i.parent.state$ と表記しているが，これは状態空間の有向全域木に採用されている遷移に対応する差分情報なので状態空間の有向全域木に格納しても差し支えない．

後半の条件分岐では，S'_{dst} と S_{dst} が同一の状態であった場合，4.6.1 節および 4.6.2 節の方針に則って状態空間の有向全域木 *spTree* に採用される遷移を更新している．

Require: 書換え前の状態 S_{src}, 書換え後の状態 S'_{dst}, 書換え後の状態との同一状態候補 S_{dst}, 書換え前後の差分情報 *diffInfo*, 状態空間の有向全域木 *spTree*
Ensure: 同型性判定結果 *isSameState*, 状態空間の有向全域木 *spTree*

```
V1 := spTree.vertex(S_src)
V2 := spTree.vertex(S_dst)
while V1 ≠ V2 do
  if V1.depth > V2.depth then
    diffInfo := diffInfo △ (V1.state △ V1.parent.state)
    V1 := V1.parent
  else if V1.depth < V2.depth then
    diffInfo := diffInfo △ (V2.state △ V2.parent.state)
    V2 := V2.parent
  else
    diffInfo := diffInfo △ (V1.state △ V1.parent.state) △ (V2.state △ V2.parent.state)
    V1 := V1.parent
    V2 := V2.parent
  end if
end while
if diffInfo = ∅ then
  isSameState := True
  V_src = spTree.vertex(S_src)
  V_dst = spTree.vertex(S_dst)
  if V_src.depth ≠ V_dst.depth then
    if V_src.depth > V_dst.depth then
      V_src.parent := V_dst
    else
      V_dst.parent := V_src
    end if
  end if
else
  isSameState := False
end if
return isSameState, spTree
```

図 4.14　基本閉路長削減アルゴリズム

第 5 章

実験

この章では，第 4 章で提案した基本閉路長削減手法を実際にグラフ書換え系モデル検査器 SLIM[4] に実装し，状態空間の基本有向全域木を利用した場合と比較することで，基本閉路長削減手法によってどの程度基本閉路の削減が見込めるか，様々なモデルに対して一定の効果が見込むことができるのかを議論する．

5.1 SLIM および LMNtal

モデル検査器 SLIM は，グラフ書換え系モデリング言語 LMNtal[6] をモデリング言語に持つモデル検査器である．例題による検証の前に，LMNtal および SLIM について紹介する．

5.1.1 グラフ書換え系モデリング言語 LMNtal

本節では，グラフ書換え系モデル言語である LMNtal に関して述べる．本節の内容は文献 [6] に基づいている．

構成要素

LMNtal は階層グラフ書き換えに基づくモデル言語であり，LMNtal の基本データ構造はアトム・リンク・膜・ルールからなる．LMNtal の特徴に，階層構造をアトム・リンク・膜によって表現できる，という点が挙げられる．以下，それぞれの構成要素に関して述べる．

アトム　アトムはアトム名と m 個 $(m \geq 0)$ の順序付けられた引数からなり，m 個の引数

を持つアトムを m 価のアトムと呼ぶこともある．それぞれの引数は (自分自身または他の) アトムにつながるリンクの端点である．

リンク　リンクはリンク名を用いて表記し，同じリンク名をもつアトムの引数同士が相互接続されていることを表す．LMNtal では 5.1.1 節に述べるリンク条件なる構文条件を設けることでリンクによるアトムの接続を一対一に制限し，アトムとリンクで形成される構造が無向グラフ構造を表すようになっている．

膜　膜はアトムの多重集合を囲むことができ，囲ったものをセルと呼ぶ．膜は膜自身が他の膜の入れ子になることを許容し，アトム，リンク，膜によって構成された階層グラフ構造を LMNtal ではプロセスと呼ぶ．

ルール　ルールとはプロセスの書換え規則であり，以下の構文で表記する．

$$Head \text{ :- } Body$$

ルールはいかなる膜にも囲まれていない状態を表すグラウンド膜を含むいずれかの膜に所属する．ルール左辺の *Head* はルールの所属する階層から見た書き換えるべきプロセスのテンプレートであり，ルール右辺の *Body* は書き換え後のプロセスのテンプレートである．各膜の中にはルールは複数個あってもよく，その多重集合をルールセットと呼ぶ．

構文

図 5.1 に LMNtal の基本構文を示す．X_i はリンク名，p はアトム名，P はプロセスである．T はプロセスの書き換え規則の表現に用いるプロセステンプレートであり，局所文脈 (特定のセルの内部での文脈) を扱う機能をもつ．

また 0 は中身の無いプロセス，$p(X_1, \ldots, X_m)$ は m 価アトム，P, P はプロセスの並列合成，P は膜 P によってグループ化されたプロセス，ルール T:-T はプロセスの書き換え規則である．

LMNtal におけるプロセス内において，同じリンク名が 2 回を超えて出現してはならない．これをリンク条件と呼ぶ．また，プロセス P に 1 回だけ出現するリンク名は P の自由リンク (P の外部につながるリンク) を表し，P に出現するそれ以外のリンク名は P の局所リンクを表す．個々のルールの各リンク名は，そのルール内にちょうど 2 回出現しな

$P ::=$	0	(空)
$\mid$	$p(X_1, \ldots, X_m)(m \geq 0)$	(アトム)
$\mid$	P, P	(分子)
$\mid$	$\{P\}$	(セル)
$\mid$	T :- T	(ルール)
$T ::=$	0	(空)
$\mid$	$p(X_1, \ldots, X_m)(m \geq 0)$	(アトム)
$\mid$	T, T	(分子)
$\mid$	$\{T\}$	(セル)
$\mid$	T :- T	(ルール)
$\mid$	@p	(ルール文脈)
$\mid$	\$$p[X_1, \ldots, X_m \mid A](m \geq 0)$	(プロセス文脈)
$A ::=$	$[]$	(空)
$\mid$	$*X$	(リンク束)

図 5.1　LMNtal の基本構文

ければならない．

ルールは膜の内部に入れることが可能であり，計算の局所化に利用可能である．ただしルールの書き換え対象はそのルールが所属する膜および所属する膜内に含まれるプロセスに限定され，ルールの含まれる膜の外の内容を書き換えることはできない．

ルール文脈は膜の中の全てのルールの多重集合とマッチし，プロセス文脈は膜の中のルール以外のプロセスのうち，明示的に指定されていないもの全体とマッチする．個々のルール中の文脈の出現はいくつかの構文条件を満たさなければならない [13]．アトム，ルール文脈，プロセス文脈は，同じ名前 p を持っていても互いに無関係である．

プロセス文脈の引数は，自由リンクの出現に関する制約条件を指定するものである．ルール左辺のプロセス文脈\$$p[X_1, \ldots, X_m \mid A](m \geq 0)$ の引数 $X_1, \ldots, X_m$ は，その文脈が持っていなければならない自由リンクを指定しており，これをプロセス文脈の明示的な自由リンクという．

5.1.2　モデル検査器 SLIM

本節の内容は [4] に基づく．

LMNtal によって記述されたモデルについてモデル検査を行うモデル検査器が SLIM である．SLIM の状態空間の構築方法は，未展開の状態がなくなるまで以下の (i)〜(iv) の操作を繰り返す，というものである．本論文では割愛するが，モデル検査を行う際は下記の操作に加え，状態空間内の閉路の探索などを行う．

(i) 状態管理表に登録されている状態のうち，未展開である状態のスタックから状態を取り出す．この状態を X とする．
このとき，取り出される状態は深さ優先順に決定される．

(ii) 選ばれた状態 X に対して，その状態が持つ各ルールの適用を試み，いずれかのルールを 1 回だけ適用した結果を新しい状態の候補として一時的に保持する．

(iii) 新しい状態の候補それぞれに対し，既存の状態のいずれかと等しいか同型性判定を行う．

(iv) 同型性判定の結果，ある新しい状態の候補が既存の状態 Y と同じであることがわかった場合，ルールの適用によって X から Y に遷移するという事実を状態遷移グラフに登録し，この候補を破棄する．
もし新しい状態の候補が既存の状態のいずれとも異なる場合は，実際に新しい状態として状態遷移グラフ・状態管理表の両方に登録し，未展開である状態のスタックに格納する．

5.2　基本閉路長削減手法の実装箇所

言うまでもないことではあるが，本研究で最も着目すべき点は，基本閉路長削減手法によって深さ優先探索中に状態空間内の有向全域木を更新した結果，有向全域木と新しく発見された遷移によってできる基本閉路長を如何に短縮できるか，という点である．そこで，本実験では，状態空間に登録する各状態に新しく

(i) (生成遷移のみからなる) 状態空間の基本有向全域木における各状態の深さ

(ii) 基本閉路長削減手法によって更新されている有向全域木における現在の各状態の親

のフィールドを用意した．そして，5.1.2 節の状態展開方法のうち，ステップ (iv) で遷移 t を状態遷移グラフに登録した直後，すなわち状態空間の有向全域木と t による閉路ができた際に，

(i) 状態空間の基本有向全域木と t からなる閉路を有向全域木を遡ることで検索，未削減時の基本閉路長を算出

(ii) 基本閉路長削減手法によって更新された有向全域木と t からなる閉路を，図 4.14 のアルゴリズムのうち閉路を検索する部分のみを利用して検索，削減時の基本閉路長を算出後，条件を満たしていれば有向全域木を更新

という形で実装することによって状態空間展開時の両基本閉路長を計測し，全状態を展開し終わった際にそれぞれの有向全域木でなした基本閉路長の平均を算出，比較するという形で実験を行った．

5.3　ハノイの塔モデルでの実験結果

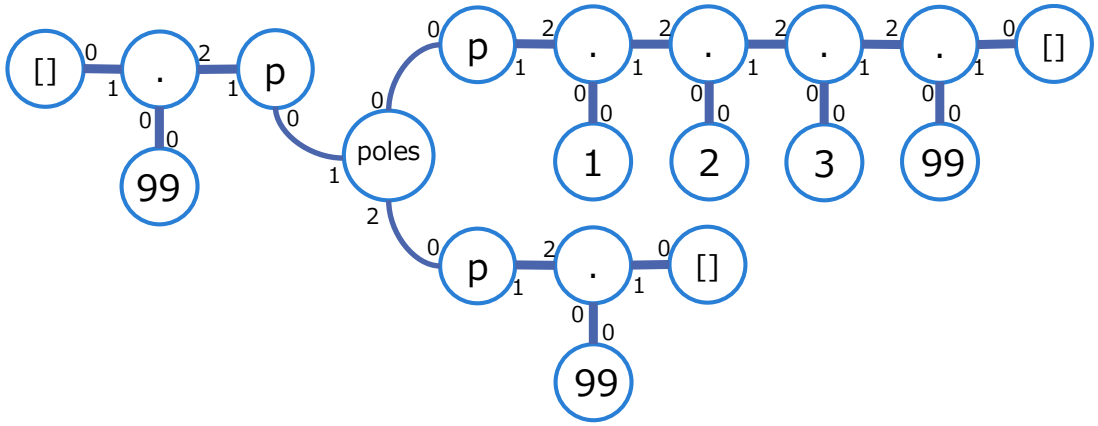

図 5.2　ハノイの塔モデルの LMNtal による表現

ハノイの塔モデルはモデルサイズを n とするとき，3 本の柱のうち 1 本に n 枚の円盤が小さい順に用意され，小さい円盤の上に大きい円盤を乗せられないという制約のもと，各柱の 1 番上の円盤を別の柱に移動させる，という操作のみで，円盤全てを別の 1 本の柱に移動させることを目指す．ハノイの塔モデルの LMNtal による表現を図 5.2 に示す．

ハノイの塔モデルでの実験結果を図 5.3 に示す．ただし，図 5.3 の系列のうち，no-reduction が状態空間の基本有向全域木と状態空間構築中に登録された遷移による基本閉

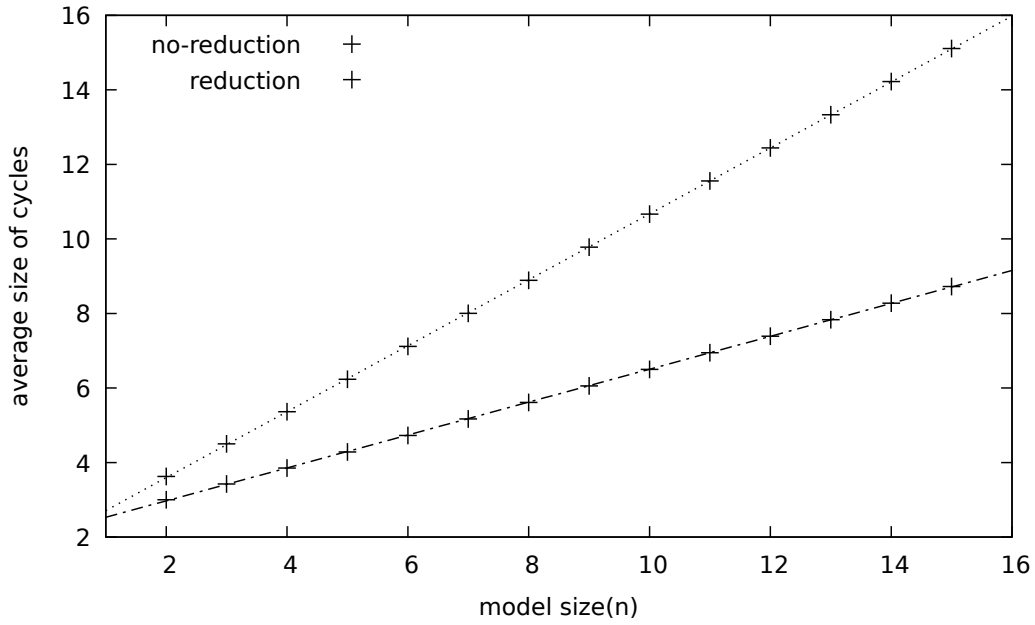

図 5.3　ハノイの塔モデルによる実験結果

路長の平均，reduction が基本閉路長削減手法を用いた基本有向全域木と状態空間構築中に登録された遷移による基本閉路長の平均である．

図 5.3 より，ハノイの塔モデルに関しては，モデルサイズに対する平均閉路長の増加が完全な線形であることがわかる．これは，モデルサイズが 1 増加すると増加する前の状態空間と同様の状態空間が 3 つできる，というハノイの塔モデルの状態空間の特殊性に由来していると考える．また，基本閉路長削減手法によって，基本閉路長の平均の増加を約 50% 削減できていることがわかる．これは，状態空間の基本有向全域木を利用した際に基本閉路長が大きくなる遷移のおよそ半分を，基本閉路長削減手法によって基本閉路長を大きく削減した結果であると考える．

5.4　食事する哲学者モデルでの実験結果

食事する哲学者モデルは並列処理およびデッドロックに関するモデルである．モデルサイズを n としたとき，円形の食卓に n 本のフォークが置いてあり，それぞれのフォークの間に計 n 人の哲学者が座っている．それぞれの哲学者は

- 両手にフォークを持っていない状態で，右手に自身から見て右側にあるフォークを

持つ．

- 右手のみにフォークを持っている状態で，左手に自身から見て左側にあるフォークを持つ．
- 両手にフォークを持っている状態で食事を行い，両手のフォークを食卓に置く．

の 3 種類の行動を行うが，哲学者全員が右手にフォークを持ってしまうと誰も左手にフォークを持つ事ができず，デッドロックに陥る．食事する哲学者モデルの LMNtal による表現を図 5.4 に示す．

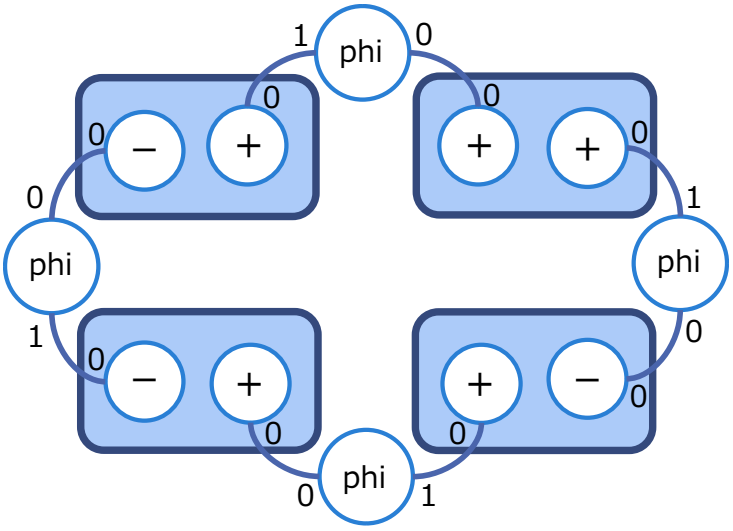

図 5.4　食事する哲学者モデルの LMNtal による表現

食事する哲学者モデルでの実験結果を図 5.5 に示す．ハノイの塔モデルと同様，図 5.5 の系列のうち，no-reduction が状態空間の基本有向全域木と状態空間構築中に登録された遷移による基本閉路長の平均，reduction が基本閉路長削減手法を用いた基本有向全域木と状態空間構築中に登録された遷移による基本閉路長の平均である．

図 5.5 においてまず目につくのが，状態空間の基本全域木を用いた際に，9 以上の奇数のサイズでその前後に比べて基本閉路長の平均が大きくなっていること，および n が 15 から 17 の時の基本閉路長の平均の増加が著しいことである．これはモデルにおける対称性がモデルサイズが奇数の時に偶数の時と比較して相対的に減少すること，状態空間の基本全域木を大きく遡る際の基本閉路長がモデルサイズに比例して大きくなることなど理由として考えられる．

一方，基本閉路長削減手法によって状態空間の有向全域木を更新しながら状態空間を展開した際は，モデルサイズに対する基本閉路長の平均の増加が相対的に非常に小さい．これは，食事する哲学者モデルにおいては，4.5 節で仮定したヒューリスティックな仮定が正しいことを証明していると考える．

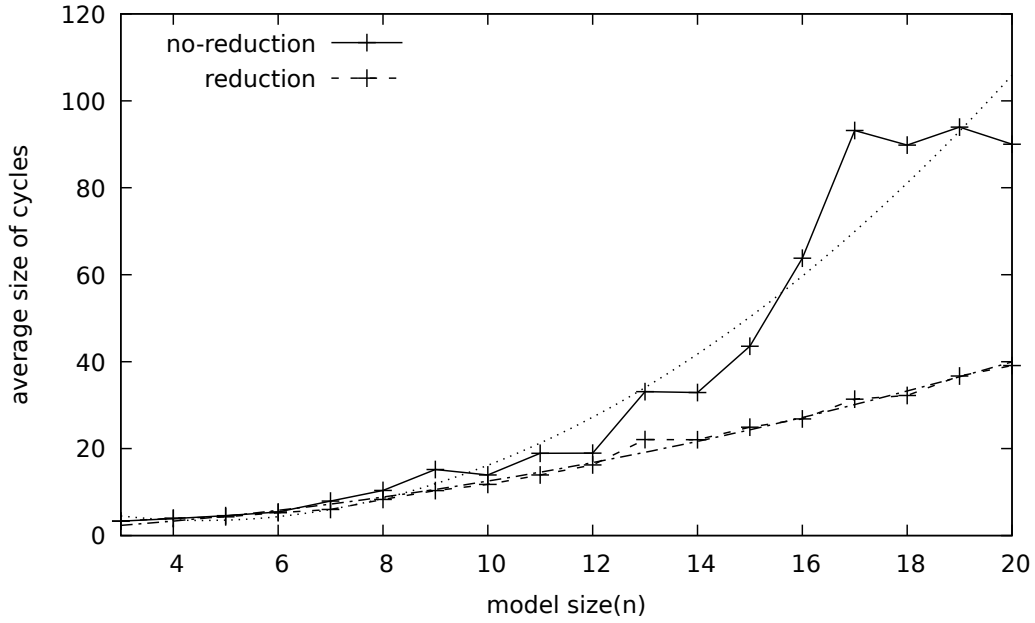

図 5.5　食事する哲学者モデルによる実験結果

5.5　SLIM ベンチマークプログラムでの実験結果

最後に，SLIM に付属しているモデル検査のベンチマークプログラムのそれぞれに対して，状態空間の基本有向全域木を利用した際の基本閉路長の平均と基本閉路長削減手法を用いた際の基本閉路長の平均を計測することで，様々なモデルの状態空間に対して基本閉路長削減手法が有向であるかを検証した．結果を表 5.1 に示す．表 5.1 の改善効率は，状態空間の基本有向全域木を利用した際の基本閉路長の平均に比べて，基本閉路長削減手法を用いた際の基本閉路長の平均が何 % 削減されているかを示す．

表 5.1 から，一部のモデルの状態数および閉路数が非常に小さい，改善効率が 5% 以下のものも数モデルあるなど，一定の不満はあるものの，様々なモデルの状態空間に対して一定の基本閉路長の削減が得られると結論付けることができる．

モデル名	状態数	遷移数	閉路数	平均基本閉路長		改善効率
				適用前	適用後	
abp	28	34	7	6.0000	5.7143	4.762%
bakery	610	1081	472	14.9958	7.6525	48.969%
barber	904	2898	1995	12.8802	8.1429	36.780%
bubble	5040	15120	10081	14.4988	11.8514	18.259%
byzantine	420	1027	608	9.2303	6.4868	29.723%
cpn	3329	12825	9497	18.4810	12.0182	34.970%
dec	252	630	379	10.8654	7.7573	28.605%
del	1024	3840	2817	10.1789	9.4633	7.030%
dtp	24	27	4	4.5000	4.0000	11.111%
edfs	7427	8131	705	14.3532	13.3887	6.720%
elevator	1398	3409	2012	12.4056	11.8280	4.656%
firewire	2825	9474	6650	16.2186	10.5926	34.689%
gcd	9724	32013	22290	11.3546	9.0642	20.172%
hl_ambient	1168	5312	4145	9.8514	8.7542	11.138%
hl_lambda	940	2964	2025	10.6074	8.0425	24.180%
hl_path	4111	18447	14337	6.8211	6.8211	0.000%
hl_ram_sim	1217	2273	1057	154.0000	4.0000	97.403%
hl_structcmp	62	111	50	7.6000	5.0000	34.211%
hl_unionfind	2100	5697	3598	9.7043	7.8210	19.407%
jsp	386	449	64	7.6875	5.8750	23.577%
knight	508493	678114	169622	26.1284	24.4772	6.320%
lambda	940	2964	2025	10.6133	7.9872	24.743%
lcm	2578	4839	2262	14.2153	7.1256	49.874%
lring	86	110	25	5.8000	5.4000	6.897%
mutex	53	149	97	5.6289	5.0722	9.890%
peterson	115	196	82	13.3659	7.1463	46.533%
phi	79	300	222	5.8423	5.4189	7.247%
phiM	1370	6382	5013	10.1420	8.6100	15.106%
qlock	134	257	124	6.7742	5.7500	15.119%
rabbit	5102	5309	208	21.6538	21.0769	2.664%
rms	2105	2486	382	5.9005	5.7225	3.017%
sieve	332	792	461	12.6681	7.3623	41.883%
sstd	708	1539	832	11.7248	8.4784	27.688%
swp	577	1530	954	17.8585	8.7327	51.101%
swp_simple	44	78	35	6.0000	5.4286	9.523%

表 5.1　ベンチマーク内の各モデルでの実行結果

第 6 章

まとめと今後の課題

6.1　まとめ

本論文では，グラフ書換え系モデル検査器へ差分適用グラフ同型性判定手法を導入する過程において，状態展開時の差分情報の処理量の改善を行う基本閉路長削減手法の提案を行った．また，基本閉路長削減手法を SLIM に実装し，評価を行い，基本閉路長削減手法によってモデルサイズの増加に対する基本閉路長の増加を抑制できること，および様々なモデルの状態空間に対して一定の基本閉路長の削減が得られることを示した．

6.2　今後の課題

6.2.1　効率的な差分情報同士の対称差の計算方法の確立

本研究は状態展開時の差分情報の処理量の削減には成功した．しかし，書換え後のグラフと同一状態候補の一意表現の対称差を計算する処理において，最も時間計算量を左右するのは，差分情報と差分情報の対称差の計算の時間計算量である．計算量の改善のためには効率的な集合同士の対称差の計算手法が必要であり，この計算手法の確立は決して簡単ではないものの，差分適用グラフ同型性アルゴリズムの実用化においては非常に重要である．

6.2.2　基本有向全域木において同一の深さである状態同士の遷移の状態空間の有向全域木への採用

本研究で提案した基本経路長削減手法においては，ループを生む可能性があるため，基本有向全域木において同一の深さである状態同士の遷移の状態空間の全域木への採用を行わなかった．しかし，同一の深さの状態同士の遷移が多い状態空間に対しては提案した基本経路長削減手法は基本閉路長の大きな削減は見込めない．

そこで，限りなく少ない処理でループの可能性を検出するなどの対策を行いながら，同一の状態の深さの状態同士の遷移を全域木に採用することができれば，本研究で提案した手法より削減効率を上げることができる可能性がある．

謝辞

本研究を進めるにあたり様々な方の指導，助言をいただきました．まず，ご指導を賜わった上田和紀教授に深く感謝致します．次に，このような不甲斐ない私と優しく接してくれました，上田研究室 22 期の同期，また，先輩として接してくれました，上田研究室 23 期・24 期のメンバー皆さんに，深く感謝致します．最後に，入学から現在に至るまで，叱咤激励を私にかけてくれた家族に，深く感謝致します．

2018 年 1 月　坂爪　裕也

参考文献

[1] Aho, Alfred V., John E. Hopcroft, and Jeffrey D. Ullman. : On finding lowest common ancestors in trees, SIAM Journal on computing Vol. 5, No. 1, pp.115-132, 1976.

[2] Hartke, S. G. and Radcliffe, A. : McKay's Canonical Graph Labeling Algorithm, Communicating Mathematics, **479** Contemporary Mathematics, American Mathematical Society, 2009, pp. 99–111.

[3] 広戸康平：LMNtal データ構造のハッシュコード化と同型性判定, 早稲田大学理工学部, 卒業論文, 2007.

[4] 堀泰祐, 佐々木隆之, 岡部亮, 村山敬, 上田和紀：LMNtal に基づくモデル検査器, 日本ソフトウェア科学会第 25 回大会論文集, 2008.

[5] 茨木 俊秀, 永持 仁, 石井 利昌：グラフ理論：連結構造とその応用, 朝倉書店, 2010.

[6] 乾敦行, 工藤晋太郎, 原耕司, 水野謙, 加藤紀夫, 上田和紀：階層グラフ書換えモデルに基づく統合プログラミング言語 LMNtal, コンピュータソフトウェア, Vol. 25, No. 1, pp. 124-150, 2008.

[7] J.Gasteiger, T.Engel 編 船津公人 監訳 船津公人 佐藤寛子 増井秀行 訳: ケモインフォマティックス—予測と設計のための化学情報学（丸善株式会社）, 2005.

[8] 川端聡基：並列モデル検査器 SLIM の状態空間削減手法および最適化問題向け状態空間構築手法, 早稲田大学基幹理工学研究科, 修士論文, 2011.

[9] McKay, B. D. : Practical Graph Isomorphism, Congr. Numer **30**, 1981, pp. 45–87.

[10] Miyazaki, T. : The complexity of McKay's canonical labeling algorithm, Groups and Computation, II (New Brunswick, NJ, 1995), DIMACS Ser. Discrete Math. Theoret. Comput. Sci., **28**. Amer. Math. Soc,. Providence, RI, 1997, pp. 239–256.

[11] 宮原 和大, 上田 和紀：グラフ書換え系のための効率的なグラフ正規化手法, コン

ピュータソフトウェア,Vol. 33, No. 1, pp. 126-149, 2016.

[12] 宮原 和大 : グラフ書換え系におけるグラフ構造の効率的な同型性判定手法, 早稲田大学基幹理工学部, 修士論文, 2013.

[13] Kazunori Ueda. LMNtal as a hierarchical logic programming language. *Theor.Comput. Sci.*, Vol. 410, No. 46, pp. 4784-4800, 2009.

[14] 吉田健人 : 並列グラフ書換え系モデル検査器 SLIM の状態空間圧縮手法の実装とその評価, 早稲田大学基幹理工学研究科, 修士論文, 2015.